НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
Кафедра обчислювальної техніки

КУРСОВА РОБОТА

з дисципліни «Паралельні та розподілені обчислення»

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

Студентки 3 курсу групи ІО-01
напряму підготовки 050102
«Комп'ютерна інженерія»
Наумової Х.С.

Керівник доцент Корочкін О.В.

Національна оцінка _____
Кількість балів: _____
Оцінка: ECTS _____

Члени комісії _____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)
_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)
_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали

Київ - 2013 рік

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) інформатики та обчислювальної техніки
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Напрям підготовки 6.050102 «Комп'ютерна інженерія»

З А В Д А Н Н Я

НА КУРСОВУ РОБОТУ СТУДЕНТУ

Наумовій Христині Сергіївні

1. Тема роботи «Розробка програмного забезпечення для паралельних комп'ютерних систем»
керівник роботи Корочкін Олександр Володимирович к.т.н., доцент
2. Строк подання студентом роботи 11 травня 2013 р.
3. Вихідні дані до роботи
   - порівняння реалізації атомік-змінних в мовах і бібліотеках паралельного програмування
   - математична задача A=(B*MO)(α*MX+MY * MT)
   - структури ПКС ОП та ПКС ЛП
   - мови програмування: Java, C++
   - бібліотека MPI
   - засоби організації взаємодії процесів: м'ютекси.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   - порівняння реалізації атомік-змінних в мовах і бібліотеках паралельного програмування
   - 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.

6. Дата видачі завдання ____6.03.2013_____

КАЛЕНДАРНИЙ ПЛАН

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

Студент _____ Наумова Х.С.

(підпис)

Керівник роботи _____ Корочкін О.В.

(підпис)

ЗМІСТ

ЗМІСТ 4
РОЗДІЛ 1. Огляд атомік-змінних 5
Вступ 5
1.1 Атомарні змінні в мові програмування Java 6
1.2 Атомарні змінні в мові програмування C#. 7
1.3 Атомарні змінні в мові програмування Ada95. 11
1.4 Атомарні змінні в бібліотеці OpenMP. 11
1.5 Атомарні змінні в бібліотеці Win 32 12
1.6 Порівняння реалізації атомік-змінних в мовах та бібліотеках 13
1.7 Висновки до Розділу 1 13
РОЗДІЛ 2. Розробка ПЗ для ПКС з СП 15
2.1 Рішення початкової математичної задачі 15
2.2 Дослідження ефективності розробленого ПЗ на реальній Р-ядерної ПКС 19
2.3 Висновки до розділу 2 21
РОЗДІЛ 3. Розробка ПЗ для ПКС з ЛП 23
3.1 Розробка алгоритмів процесів. 23
3.2 Розробка схеми взаємодії процесів 24
3.3 Розробка програми 24
3.4 Дослідження ефективності розробленого ПЗ на реальній Р-ядерної ПКС 25
3.5 Висновки до розділу 3 27
РОЗДІЛ 4. Основні висновки та результати роботи 29
РОЗДІЛ 5. Список використаної літератури 30
РОЗДІЛ 6. Додатки 32
Додаток А. Структурна схема ПКС з ЗП 32
Додаток Б. Структурна схема ПКС з ЛП 32
Додаток В. Алгоритм основної програми для ПКС з ЗП 32
Додаток Г. Алгоритм процесів в програмі для ПКС з ЗП 32
Додаток Д. Алгоритм основної програми для ПКС з ЛП 32
Додаток Е. Алгоритм процесів у програмі для ПКС з ЛП 32
Додаток Ж. Лістінг програми для ПКС з ЗП 32
Додаток З. Лістінг програми для ПКС з ЛП. 32

РОЗДІЛ 1. Огляд атомік-змінних

Вступ

Найменш ресурсоємними засобами забезпечення атомарного виконання операцій є атомарні змінні [1]. Атомарні операції - операції, що виконуються як єдине ціле або не виконуються зовсім. В багатопроцесрних комп'ютерах та багатозадачних операційних системах дуже велике значення має атмоарність операцій, так як доступ до ресурсів також обов'язково повинен бути атомарним. Атомарність досягається за допомогою таких механізмів міжпрограмної взаємодії як семафори та м’ютекси. Основною особливістю атомарної змінної є те, що як тільки якийсь процес починає їх зчитувати або записувати в них, то інший процес не може його перервати і виконатися в середині. Отже, ніщо не може розбити процес доступу до атомарної змінної на декілька частин.

Атомарна операція функціонально еквівалентна «критичній секції» (блокуванню) [2], разом з тим треба докласти зусиль, щоб не понести значні втрати порівняно з прямим використанням атомарних операцій, пряму підтримку якої пропонують багато комп'ютерних архітектур. Щоб покращити продуктивність програми, часто є доброю ідеєю замінити критичні секції на атомарні операції для неблокуючої синхронізації. Проте значне покращення швидкодії не гарантоване, і неблокуючі алгоритми можуть стати просто не вартими зусиль на втілення.

Більшість сучасних процесорів має інструкції, які можна застосувати для реалізації блокування і неблокуючих алгоритмів. Для однопроцесорних систем також досить мати можливість тимчасово забороняти переривання, щоб бути певним, що виконуваний процес не переключить контекст.

Наступні інструкції [1] прямо використовуються компіляторами і операційними системами, але також можуть бути абстраговані і представлені як байт-код або бібліотечні функції в мовах високого рівня:

1 Atomic read and write (атомарні зчитування та запис)

2 Test-and-set (перевірити-і-встановити)

3 Fetch-and-add (вибрати-і-додати)

4 Compare-and-swap (порівняти-і-переставити)

5 Load-Link/Store-Conditional (Завантажити-зв'язати/Зберегти-умовно)

Багато з цих примітивів можуть бути реалізовані в термінах інших.

1.1 Атомарні змінні в мові програмування Java.

Інструментарій для реалізації атомарних змінних та операцій в мові програмування Java представлений класами та методами цих класів з пакету java.util.concurrent.atomic [3]. Приведено загальний огляд класів пакету java.util.concurrent.atomic та їх опис (таблиця 1.1).

Java.util.concurrent.atomic – невеликий набір інструментів класів, які підтримують не блокуючі потоково-безпечні операції над атомарними змінними [4]. Класи в цьому пакеті розширяють ідею змінних, полів та масивів позначених модифікатором volatile, які також забезпечують умовну атомарну операцію оновлення у вигляді:

boolean compareAndSet(expectedValue, updateValue);

Цей метод атомарно встановлює змінну updateValue, якщо вона в даний час займає expectedValue, та повертає значення true при успішному виконанні. Класи в цьому пакеті також містять методи для отримання і безумовного встановлення значення, так само як слабша умовна атомарна операція оновлення weakCompareAndSet.

Таблиця 1.1. Огляд класів пакету java.util.concurrent.atomic

AtomicBoolean	Змінна типу boolean, що може оновлюватись атомарно.
AtomicInteger	Змінна типу int, що може оновлюватись атомарно.
AtomicIntegerArray	Масив типу int, елементи якого можуть оновлюватись атомарно.
AtomicIntegerFieldUpdater<T>	Допоміжний клас оснований на рефлексії що дозволяє атомарне оновлення призначених полів помічених як volatile int в призначеному

	класі.
AtomicLong	Змінна типу long, що може оновлюватись атомарно.
AtomicLongArray	Масив типу int, елементи якого можуть оновлюватись атомарно.
AtomicLongFieldUpdater<T>	Допоміжний клас оснований на рефлексії що дозволяє атомарне оновлення призначених полів помічених як volatile long в призначеному класі.
AtomicMarkableReference<V>	Клас зберігає посилання на об’єкт разом з міткою, що можуть бути оновлені атомарно.
AtomicReference<V>	Посилання на об’єкт, що може оновлюватись атомарно.
AtomicReferenceArray<E>	Масив посилань на об’єкти, елементи якого можуть оновлюватись атомарно.
AtomicReferenceFieldUpdater<T,V>	Допоміжний клас оснований на рефлексії що дозволяє атомарне оновлення призначених полів помічених як volatile reference в призначеному класі.
AtomicStampedReference<V>	Клас зберігає посилання на об’єкт разом з «штампом» типу integer, що можуть бути оновлені атомарно.

Специфікації цих методів дозволяють використовувати реалізації ефективних атомарних команд на машинному рівні, які доступні на сучасних процесорах.

Екземпляри класів AtomicBoolean, AtomicInteger, AtomicLong, і AtomicReference, забезпечують доступ і поновлення в одну змінну відповідного типу. Також кожен клас забезпечує відповідні допоміжні методи відповідного типу. Наприклад, клас AtomicLong та AtomigInteger надають методи атомарної інкрементації.

1.2 Атомарні змінні в мові програмування C#.

Інструментарій для роботи з атомарними змінними в цій мові представлений класом System.Threading.Interlocked [5, 6]. Методи цього

класу, що представлені в таблиці 1.2, не захищені від помилок, які можуть виникнути при перемиканні контекстів планувальником потоків в той час, коли потік оновлює змінну, а вона може бути доступна іншим потокам, або коли одночасно виконуються два потоки на різних процесорах. Методи цього класу не генерують виключень.

Методи Increment і Decrement збільшують або зменшують значення змінної і зберігають результат в одній операції. На більшості комп'ютерів збільшення значення змінної не є атомарної операцією, а вимагає наступних кроків:

1. Завантажити значення з змінної примірника в регістр.
2. Збільшити або зменшити значення.
3. Зберегти значення в змінній примірника.

Якщо не використовувати методи Increment і Decrement, потік може бути перерваний після виконання перших двох кроків. Потім інший потік може виконати всі три кроки. Коли перший потік продовжить виконання, він переписує значення в змінну екземпляру, і ефект збільшення або зменшення значення, виконаного іншим потоком , втрачається. Метод Exchange обмінює значення заданих змінних на рівні атомарних операцій. Метод CompareExchange поєднує в собі дві операції: порівнювання двох значень і збереження третього значення в одній із змінних, яке залежить від результату порівняння. Операції порівняння і обміну виконуються як атомарні.

Таблиця 1.2. Методи класу System.Threading.Interlocked

Add(Int32/64, Int32/64)	Додає два 32/64-розрядних цілих числа і замінює перше число на суму в одній атомарній операції.
CompareExchange(Double, Double, Double)	Порівнює два числа з плаваючою комою з подвійною точністю на рівність і, якщо

	вони рівні, замінює одне зі значень.
CompareExchange(Int32/64, Int32/64, Int32/64)	Порівнює два 32/64-розрядних знакових цілих числа на рівність і, якщо вони рівні, замінює одне зі значень.
CompareExchange(IntPtr, IntPtr, IntPtr)	Порівнює два залежних від платформи обробника або покажчика на рівність і, якщо вони рівні, замінює одне зі значень.
CompareExchange(Object, Object, Object)	Порівнює два об'єкти на рівноправність і, якщо вони рівні, замінює один з них.
CompareExchange(Single, Single, Single)	Порівнює два числа з плаваючою комою зі звичайною точністю на рівність і, якщо вони рівні, замінює одне зі значень.
CompareExchange<T>(T, T, T)	Порівнює два екземпляра зазначеного посилального типу T на рівність і, якщо це так, замінює один з них.
Decrement(Int32/64)	Зменшує значення заданої змінної і зберігає результат - як атомарна операція.
Exchange(Double, Double)	Задає число з плаваючою комою з подвійною точністю вказаним значенням - як атомарна операція, і повертає вихідне значення.
Exchange(Int32/64, Int32/64)	Задає 32/64-розрядне знакове ціле число заданим значенням - як атомарна операція, і повертає вихідне значення.

Exchange(IntPtr, IntPtr)	Задає покажчик або обробник, залежний від платформи - як атомарна операція, і повертає посилання на вихідне значення.
Exchange(Object, Object)	Задає об'єкт вказаним значенням - як атомарна операція, і повертає посилання на вихідний об'єкт.
Exchange(Single, Single)	Задає число з плаваючою комою зі звичайною точністю вказаним значенням - як атомарна операція, і повертає вихідне значення.
Exchange<T>(T, T)	Задає змінну зазначеного типу T в значення і повертає вихідне значення.
Increment(Int32/64)	Збільшує значення заданої змінної і зберігає результат - як атомарна операція.
MemoryBarrier	Синхронізувати доступ до пам'яті таким чином: Процесор, який виконує поточний потік не може змінювати порядок інструкцій таким чином, що доступ до пам'яті до того, як виклик MemoryBarrier виконується після отримання доступу до пам'яті, шляхом виклику MemoryBarrier.
Read	Повертає 64-бітове значення, завантажене у вигляді атомарної операції.

1.3 Атомарні змінні в мові програмування Ada95.

Директиви компілятора Atomic та Atomic_Components надають можливість роботи з атомарними змінними [7]. Ці директиви забезпечують неперервність операцій зчитування/запису для всіх вказаних в них об’єктах. Такі об’єкти називаються атомарними, а операції над ними виконуються лише послідовно.

Зазвичай директиви використовуються в контексті взаємодії з апаратними регістрами, які повинні бути написані атомарно і де компілятор не повинен змінювати порядок чи припиняти будь які надлишкові операції зчитування чи запису. В деяких випадках вони можуть бути використані для управління потоками, але для взаємодії потоків в Ada, безпечніше використовувати захищені типи або інші примітиви синхронізації.

Використання атомарних змінних вказані в прикладі 1.1.

Приклад 1.1. Атомарні змінні, Ada95

```
Array_Size  : Positive;
pragma Atomic   (Array_Size);
Store_Array is array (1..Array_Size)  of Integer;
pragma Atomic_Components   (Store_Array);
```

1.4 Атомарні змінні в бібліотеці OpenMP.

В даній бібліотеці механізм роботи з атомарними змінними надає директива omp atomic [8]. Дана директива відноситься до оператору присвоєння, що йде безпосередньо за нею, гарантуючи коректну роботу з спільною змінною, що стоїть в його лівій частині. Під час виконання оператора, блокується доступ до цієї змінної всім запущеним на даний момент потокам, окрім потоку, який виконує операцію. Атомарною є тільки робота зі змінною в лівій частині оператора присвоєння, при цьому обчислення в правій частині не обов’язково мають бути атомарними [9]. Приклад 1.2 демонструє використання директиви atomic.

Приклад 1.2. Використання atomic, OpenMP

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
 int count = 0;
#pragma omp parallel
 {
#pragma omp atomic
 count++;
 }
 printf("Число потоків: %d\n", count);
}
```

1.5 Атомарні змінні в бібліотеці Win 32

Засоби роботи з атомарними змінними в бібліотеці Win32, дуже схожі з засобами мови C#. Атомарні операції представляють собою методи, що починаються з префіксу Interlocked [10]. Суть цих операцій дуже проста – виконати декілька простих операцій атомарно, за один крок, щоб їх виконання не було перерване іншим потоком.

Наприклад, функції InterlockedIncrement та InterlockedDecrement виконують атомарно одразу дві операції: змінюють лічильник на одиницю та зрівнюють результат з нулем. Якби ці функції не дозволяли б виконувати операцію порівняння, то вони б не мали такої цінності, так як довелося б знову зчитувати значення лічильника з пам’яті, а в цей час його міг би змінити інший потік. Це загальна властивість всіх Interlocked функцій. Переваги атомарних операцій перед критичними секціями чи м’ютексами в тому, що вони підтримуються на рівні процесорних команд, що з точки зору продуктивності та затрат ресурсів найефективніший інструмент синхронізації. Приклад використання атомарних функцій у Win32 наведено в прикладі 1.3.

Приклад 1.3. Атомарні функції, Win32

```
void CReferenceCounter::Release( void ) {
        if( !::IntrlockedDecrement( &m_nReferences ) ) {
        delete this;
        }
}
```

1.6 Порівняння реалізації атомік-змінних в мовах та бібліотеках

Загалом механізм реалізації у всіх схожий, але має різні підходи.

В Java та C# використовуються відповідні класи та їх методи. В OpenMP та Ada95 – директиви #pragma Atomic та #pragma omp atomic. У Win32 використовуються функції з префіксом Interlocked. Переваги використання атомарних операцій перед їх аналогами в тому, що ці операції підтримуються на рівні процесорних команд, що дає високу продуктивність та оптимальне використання ресурсів.

1.7 Висновки до Розділу 1

1. Виконано аналіз засобів реалізації атомарних змінних мови Java. Показано, що вони базуються на використанні класів з пакету java.util.concurrent.atomic, який дозволяє виконувати атомарні операції дотримуючись принципу CAS(Compare-and-Swap).

2. На основі аналізу реалізації атомарних змінних в мові програмування C# можна зробити висновок, що ця мова має повний механізм роботи з атомарними змінними, а саме атомарні операції, що надаються з класу System.Threading.Interlocked. Клас надає широкий спектр функцій, серед яких атомарний інкремент/декремент, атомарна заміна чи атомарне зчитування.

3. Проаналізовано мову програмування Ada95 на реалізацію атомарних змінних. Виявлено, що вона надається за допомогою директив Atomic та Atomic_Components. В свою чергу операції над змінними,

описаними за допомогою цих директив, виконуються лише атомарно, тобто послідовно.

4. Виконано аналіз бібліотеки паралельного програмування OpenMP, щодо атомарних змінних. Показано, що їх реалізація, подібно мові програмування Ada, надається за допомогою використання директив omp atomic. Завдяки використанню даного механізму, змінна блокується для всіх потоків, окрім того, який виконує над змінною потрібні операції.

5. На основі аналізу бібліотеки паралельного програмування Win32 можна зробити висновок, що атомарні операції надаються за рахунок функцій, що починаються з префіксу Interlocked, та виконують над змінними атомарні операції за один крок, не даючи перервати цей процес іншим потокам.

РОЗДІЛ 2. Розробка ПЗ для ПКС з СП

У цьому розділі буде розглянуто та проаналізовано поставлену математичну задачу, розроблено алгоритм рішення та схему взаємодії потоків для системі з спільною пам’яттю, а на основі вищезазначених досліджень буде створена паралельна програма.

2.1 Рішення початкової математичної задачі

Матаматична задача, для якої необхідно створити паралельну програму: $A = (MO*B) * (\alpha*MX + MY*MT)$.

Перш за все розроблюється математичний алгоритм, аналізується наявність спільних ресурсів. \

1)$T_H = (B*MO_H)$
2)$A_H = T*(MY*MT_H + \alpha*MX)$
Спільні ресурси: MY,MX,B,α

Наступним етапом є створення алгоритму для кожного паралельного процесу із зазначенням точок синхронізації та критичних ділянок.

Т1		КУ,ТС
1.	Чекати сигналу про читання від Тр	WP,1
2.	Створити копії alpha, B, MY, MX	КУ
3.	$T_H = (B*MO_H)$	
4.	Сигнал про завершення рахунку Тн для всіх	S1,1
5.	Чекати сигналу про про завершення рахунку Т	W2..P ,1
6.	Створити копію Т	КУ
7.	$A_H = T*(MY*MT_H + \alpha*MX)$	
8.	Чекати сигналу про про завершення рахунку А від всіх	W2..P,2
9.	Виведення А	

Ti (1 < i < P)	КУ,ТС
1. Чекати сигналу про читання від T2, T4	W2,1;W4,1
2. Створити копії alpha, B, MY, MX	КУ
3. $T_H = (B*MO_H)$	
4. Сигнал про завершення рахунку Тн для всіх	Si,2
5. Чекати сигналу про про завершення рахунку T	W1..i-1,2;Wi+1..P,2
6. Створити копію T	КУ
7. $A_H = T*(MY*MT_H + \alpha*MX)$	
8. Сигнал про завершення рахунку для T1	Si,3

T4	КУ,ТС
1. Читання alpha, B, MY, MX	
2. Сигнал про завершення читання для всіх	SP,1
3. Чекати сигналу про читання від T2	W2,1
4. $T_H = (B*MO_H)$	
5. Сигнал про завершення рахунку Тн для всіх	SP,2
6. Чекати сигналу про про завершення рахунку T	W1..P-1,2
7. Створити копію T	КУ
8. $A_H = T*(MY*MT_H + \alpha*MX)$	
9. Сигнал про завершення рахунку для T1	Si,3

T2	
1. Читання MT, MO	
2. Сигнал про завершення читання для всіх	SP,1
3. Чекати сигналу про читання від T4	W2,1
4. $T_H = (B*MO_H)$	
5. Сигнал про завершення рахунку Тн для всіх	SP,2
6. Чекати сигналу про про завершення рахунку T	W1..P-1,2

7. Створити копію T КУ
8. $A_H = T*(MY*MT_H + \alpha*MX)$
9. Сигнал про завершення рахунку для T1 Si,3

На підставі даного алгоритму будується схема взаємодії процесів, представлена в додатку Г «Алгоритми процесів в програмі для ПКС з СП». Як засіб синхронізації обраний механізм монітора, зображенного на рисунку 2.1, в програмі створений клас з даною назвою, який реалізує виконання задач взаємного виключення і синхронізації, представлених вище в алгоритмі.

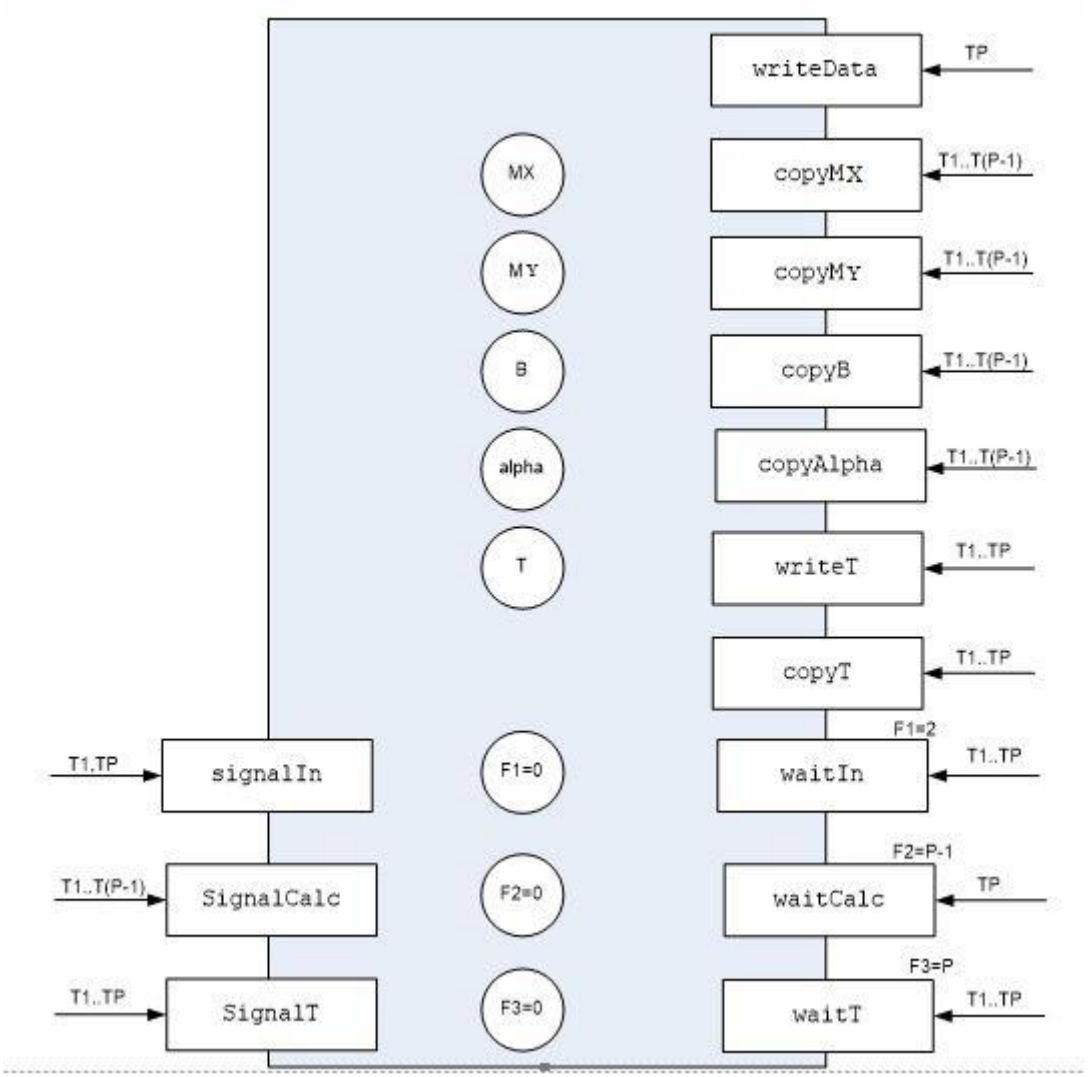


Рисунок 2.1 – Монітор

Використовуються такі методи:

- WriteData - запис в монітор CP ()

- WriteT - запис в монітор T (проміжний результат)

- CopyMX - створення локальної копії MX

- CopyMY - створення локальної копії MY

- CopyB - створення локальної копії B

- CopyAlpha - створення локальної копії alpha

- CopyT - створення локальної копії T

- WaitIn - очікування закінчення введення T2 і T4 (синхронізація по вводу). Прапор F1 спочатку встановлений в 0. При закінченні введення якихось із цих потоків, виконується операція F1 + + (в методі signalIn). Якщо F1 = 2, все потоки повідомляються про це і отримують можливість доступу до даних і до подальшого виконання програми.

- WaitCalc - очікування закінчення рахунку (синхронізація з виведення). Прапор F2 спочатку встановлений в 0. При закінченні рахунки яким або з потоків, виконується операція F2 + + (в методі SignalCalc). Якщо F1 = 5, потік (P) отримує повідомлення про це і допускається до подальшого виконання програми.

- WaitT - очікування закінчення рахунку T. Прапор F3 спочатку встановлений в 0. При закінченні рахунки яким або з потоків, виконується операція F3 + + (в методі SignalT). Якщо F1 = 6, всі потоки отримують повідомлення про цю подію і допускається до подальшого виконання програми.

- SignalIn - повідомлення від потоку про завершення їм введення даних. F1 + +.

- SignalCalc - повідомлення від потоку про завершення їм рахунки. F2+ +.

- SignalT - повідомлення від потоку про завершення їм рахунки Тн. F3 + +.

Лістинг програми розміщений у додатку Ж «Лістинг програми для ПКС з СП».

2.2 Дослідження ефективності розробленого ПЗ на реальній Р-ядерної ПКС

Для дослідження необхідно визначити час виконання розробленої програми в реальній ПКС. Тому при тестуванні послідовно використовуються 1, 2, 4, 6 процесорів, для яких визначаються час виконання Програми 1. При цьому встановлюється декілька значень розмірності векторів (матриць) N = 800, 1000, 1200. В таблиці 2.1 відображаються значення часу для різних N та Р.

Таблиця 2.1 - Час виконання програми в ПКС з СП (в мс)

N//P	T1	T2	T4	T6
800	25437	13387	7019	5332
1000	31797	17003	9090	6761
1200	38156	20964	11215	8149

Базуючись на отриманих результатах, вираховуємо значення коефіцієнтів прискорення  . У таблиці 2.2 приведено результати цих розрахунків.

Таблиця 2.2. Значения Кп для ПКС з СП

N//P	T2	T4	T6

800	1,9037	3,4201	4,7880
1000	1,8681	3,4947	5,1289
1200	1,6279	2,8852	4,1219

На основі отриманих коефіцієнтів прискориння будується графік змінення $К_П$ в залежності від P та N. Графік наведено на рис. 2.2. На осі абцис разташована розмірність векторів та матриць, ординатами зазначено коефіцієнт прискорення. Для позначення результатів різних багатопроцесорних систем використовуються різні типи ліній та маркеров.

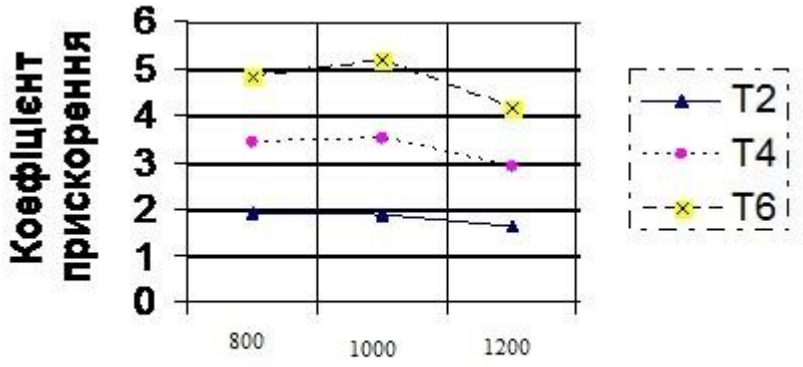


Рисунок 2.2 - Графік залежності коефіцієнта прискорення від розмірності матриць і векторів для ПКС з СП.

Розрахуємо коефіцієнти ефективності та заповнимо їми таблицю 2.3. Для розрахунку використовуються формули  .

Таблиця 2.3 - Значення Ке для програми ПКС з СП

N//P	T2	T4	T6
800	0,9518	0,8551	0,7980

1000	0,9341	0,8737	0,8548
1200	0,8139	0,7213	0,6870

На основі отриманих коефіцієнтів ефективності будується графік змінення Ке в залежності від P та N. Графік наведено на рис. 2.3. На осі абцис разташована розмірність векторів та матриць, ординатами зазначено коефіцієнт ефективності. Для позначення результатів різних багатопроцесорних систем використовуються різні типи ліній та маркеров.

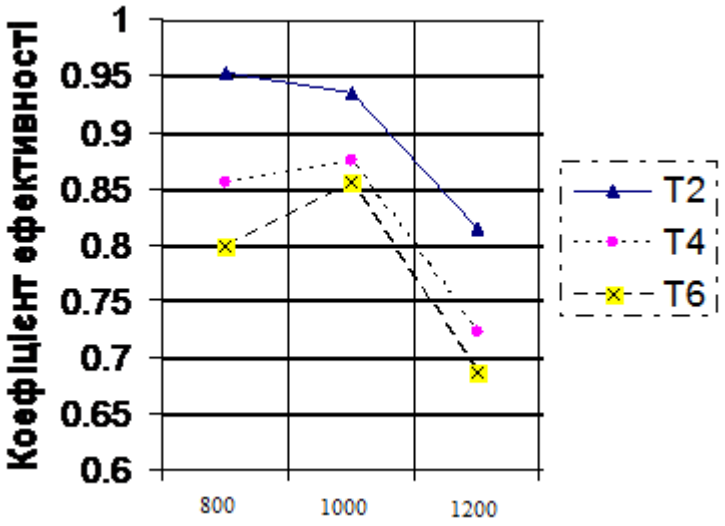


Рисунок 2.3 - Графік залежності коефіцієнта прискорення від розмірності матриць і векторів для ПКС з СП.

2.3 Висновки до розділу 2

1. У даному розділі досліджені результати тестування паралельної програми для системи з загальною пам’яттю, написаної на мові Java. Тестування провдилось для 2, 4 та 6 потоків. Для розрахунку коефіцієнтів прискорення та ефективності була протестована окремо створена послідовна програма.

2. Коефіцієнт прискорення приймає значення у проміжку від 0,6870 до 5,1289. Найвищі значення цього коефіцієнту приходяться на систему з шістьма потоками, найменше прискорення отримане для системи з двома

потоками. Характер графіків коефіцієнтів прискорення однаковий для систем з 2, 4 та 6 ядрами.

3. Коефіцієнт ефективності приймає значення у проміжку від 1,6279 до 0,9518. Найвище значення цього коефіцієнту отримане на тесті з параметрами P = 2, N = 800; найнижче – P = 6, N = 1200. Тобто, при збільшенні кількості ядер та розмірності матриць, коефіцієнт ефективності спадає.

4. Результати дослідження мають похибку, зумовлену тим, що процесори виділяються операційною системою не на монопольне використання, тобто, процесорний час може бути в будь-який час передано сторонній програмі. Чим менший час виконання програми, тим більша вірогідність виникнення досить значущої похибки.

РОЗДІЛ 3. Розробка ПЗ для ПКС з ЛП

У цьому розділі буде розглянуто та проаналізовано поставлену математичну задачу, розроблено алгоритм рішення та схему взаємодії задач, а на основі вищезазначених досліджень буде створена паралельна програма для системи з локальною пам’яттю.

Математична задача співпадає з уже розглянутою у розділі 2.1, тому розділ «Рішення початкової математичної задачі» пропущений.

3.1 Розробка алгоритмів процесів.

Програмне забезпечення є масштабованим, тобто працює на системі з будь-якою кількістю процесорів. Тому написаний алгоритм єдиний для всіх задач.

Задачі T(0)-T(P-1)

1. Якщо rank = 0, ввести дані α, B, MO, MT, MX, MY
2. Якщо rank = 0, передати всім задачам α, B, MO_H, MT_H, MX_H, MY.
3. Якщо rank !=0, прийняти α, B, MO_H, MT_H, MX_H, MY від задачі T(0).
4. Обчислення $T_H=B \cdot MO_H$.
5. Якщо rank != 0, передати T_H задачі T(0).
6. Якщо rank = 0, прийняти T_H від усіх задач.
7. Якщо rank = 0, передати T всім задачам.
8. Якщо rank != 0, прийняти T від задачі $T(0)$.
9. Обчислення $A_H=T \cdot (MY \cdot MT_H + \alpha \cdot MX_H)$.
10. Якщо rank != 0, передати A_H задачі T(0).
11. Якщо rank = 0, прийняти A_H від усіх задач.
12. Якщо rank = 0, вивести результат A.

3.2 Розробка схеми взаємодії процесів

Схема взаємодії задач приведена на рисунку 3.1.

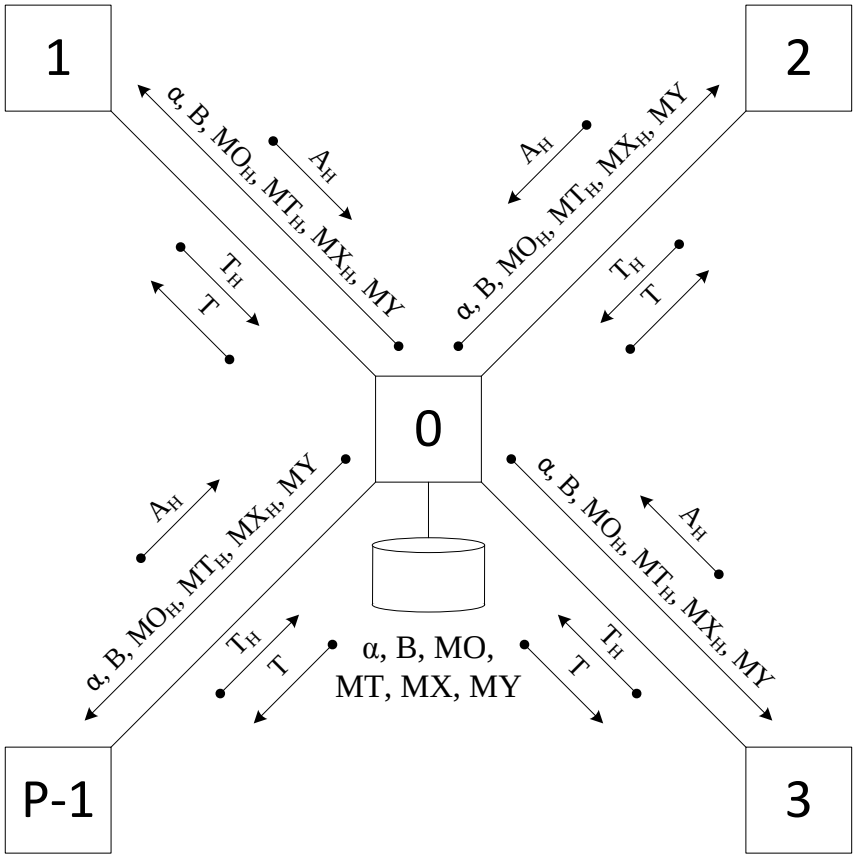


Рисунок 3.1 – Схема взаємодії задач в програмі ПРГ2

3.3 Розробка програми

Програма для ПКС ЛП складена на мові C++ з використанням бібліотеки MPI та містить чотири модулі: основний prg2.cpp та три допоміжні vector.cpp, matrix.cpp, data.cpp.

Основний модуль містить точку входу в програму main та реалізує алгоритм процесів, описаний в розділі 3.1. Для передачі і приймання використовувались функції MPI_Bcast, MPI_Scatter, MPI_Gather.

В модулях vector.cpp та matrix.cpp представлені класи Vector і Matrix, за допомогою яких створюються вектори і матриці. У цих класах визначені основні операції над елементами векторів і матриць.

Модуль data.cpp зберігає процедури, за допомогою яких формуються основні обчислення у програмі ПРГ2.

Лістинг програми наведено у додатку З.

3.4 Дослідження ефективності розробленого ПЗ на реальній Р-ядерної ПКС

Для дослідження необхідно визначити час виконання розробленої програми в реальній ПКС. Тому при тестуванні послідовно використовуються 1, 2, 4, 6 процесорів, для яких визначаються час виконання Програми 1. При цьому встановлюється декілька значень розмірності векторів (матриць) N = 800, 1000, 1200. В таблиці 3.1 відображаються значення часу для різних N та Р.

Таблиця 3.1 - Час виконання програми в ПКС з ЛП (в мс)

N//P	T1	T2	T4	T6
800	17169	8621	5236	4121
1000	28235	15234	8730	6322
1200	29472	16642	8074	6842

Базуючись на отриманих результатах, вираховуємо значення коефіцієнтів прискорення  . У таблиці 3.2 приведено результати цих розрахунків.

Таблиця 3.2. Значения Кп для ПКС з ЛП

N//P	T2	T4	T6

800	1,991	3,279	4,1662
1000	1,8534	3,234	4,466
1200	1,770	3,198	4,1219

На основі отриманих коефіцієнтів прискориння будується графік змінення $К_П$ в залежності від P та N. Графік наведено на рис. 3.2. На осі абцис разташована розмірність векторів та матриць, ординатами зазначено коефіцієнт прискорення. Для позначення результатів різних багатопроцесорних систем використовуються різні типи ліній та маркеров.

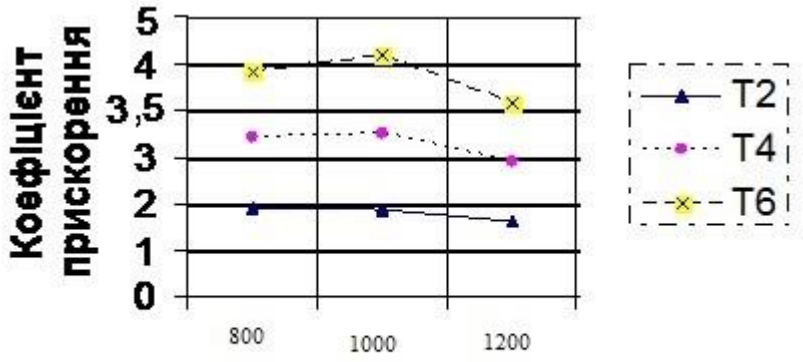


Рисунок 3.2 - Графік залежності коефіцієнта прискорення від розмірності матриць і векторів для ПКС з ЛП.

Розрахуємо коефіцієнти ефективності та заповнимо ними таблицю 3.3. Для розрахунку використовуються формули $K_{e2} = \frac{T_2}{2T_4}$, $K_{e4} = \frac{T_2}{4T_6}$, $K_{e6} = \frac{T_4}{6T_6}$.

Таблиця 3.3 - Значення Ке для програми ПКС з ЛП

N//P	T2	T4	T6

800	0,9518	0,8551	0,7980
1000	0,9341	0,8737	0,8548
1200	0,8139	0,7213	0,6870

На основі отриманих коефіцієнтів ефективності будується графік змінення Ке в залежності від P та N. Графік наведено на рис. 3.3. На осі абцис разташована розмірність векторів та матриць, ординатами зазначено коефіцієнт ефективності. Для позначення результатів різних багатопроцесорних систем використовуються різні типи ліній та маркеров.

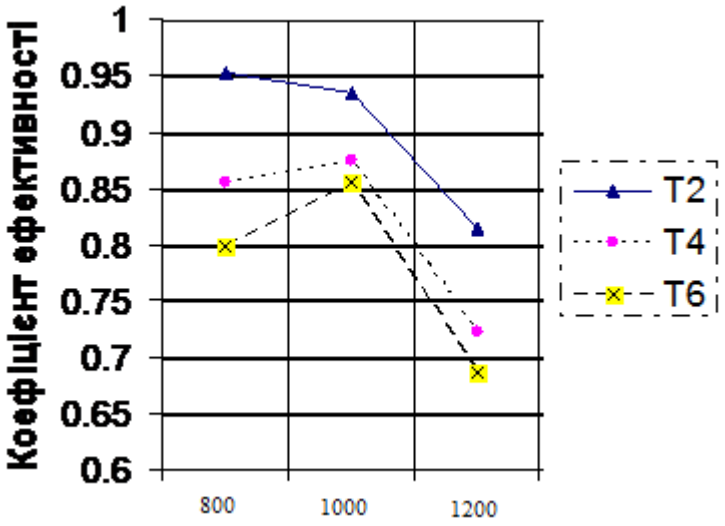


Рисунок 3.3 - Графік залежності коефіцієнта прискорення від розмірності матриць і векторів для ПКС з ЛП.

3.5 Висновки до розділу 3

1. У даному розділі досліджені результати тестування паралельної програми для системи з локальною пам’яттю, написаної на мові C++ з використанням бібліотеки MPI. Тестування проводилось для 2, 4 та 6 потоків. Для розрахунку коефіцієнтів прискорення та ефективності була протестована окремо створена послідовна програма.

2. Коефіцієнт прискорення приймає значення у проміжку від 1,770 до 4,466. Найвищі значення цього коефіцієнту приходяться на систему з

шістьма потоками, найменше прискорення отримане для системи з двома потоками. Характер графіків коефіцієнтів прискорення однаковий для систем з 2, 4 та 6 ядрами.

3. Коефіцієнт ефективності приймає значення у проміжку від 0,6870 до 0,9518. Найвище значення цього коефіцієнту отримане на тесті з параметрами P = 2, N = 800; найнижче – P = 6, N = 1200. Тобто, при збільшенні кількості ядер та розмірності матриць, коефіцієнт ефективності спадає.

4. Результати дослідження мають похибку, зумовлену тим, що процесори виділяються операційною системою не на монопольне використання, тобто, процесорний час може бути в будь-який час передано сторонній програмі. Чим менший час виконання програми, тим більша вірогідність виникнення досить значущої похибки.

РОЗДІЛ 4. Основні висновки та результати роботи

1. У даному курсовому проекті було розглянуто варіанти реалзіції атомік-змінних у вивчених протягом курсу мовах програмування і бібліотеках. Не зважаючи на велику кількість різних підходів і методів синхронізації, атомік-змінні є ефективним способом реалізації паралелізму.

2. Тестування програм для системи як з загальною пам’яттю, так і з локальною, показало передбачувані результати, а саме, зростання прискорення виконання програми при збільшенні кількості процесорів (для такої самої розмірності векторів та матриць).

3. Більш ефективною виявилась програма для системи з локальною пам’яттю, реалізована на мові C++ з використанням біліотеки MPI. Це може бути зумовлено декількома причинами. Процесори у системі з ЛП утворюють зірку, тому передача даних, що проходить паралельно у декількох напрямках зірки, на відносно невеликій кількості процесорів займає приблизно стільки ж часу, скільки отримання захищених монітором даних у програмі для системи з ЗП.

4. При тестуванні мала місце похибка, зумовлена непередбачуваним для користувача розподіленням процесорного часу операційною системою. Похибка зменшується при збільшенні розмірів вхідних даних, тобто, залежить від часу виконання програми.

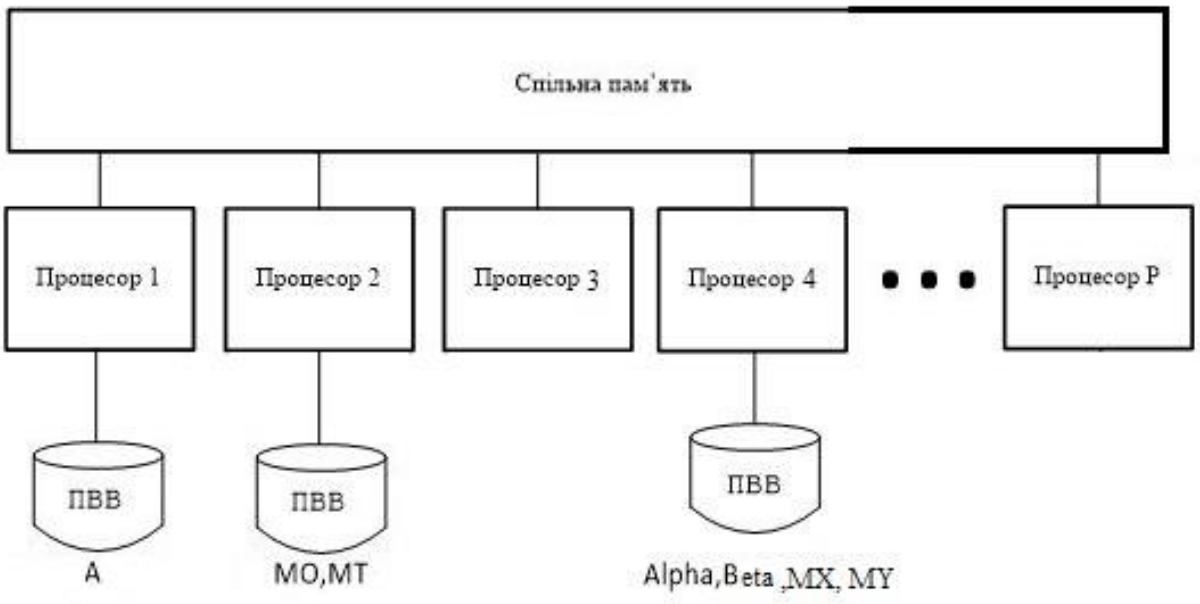
РОЗДІЛ 5. Список використаної літератури

1. Atomic типы [Електронний ресурс] – Режим доступу: http://please.noroutine.me/2011/08/atomic.html (дата звернення: 26.03.13). – Назва з екрану.
2. From beginner in Java: atomic type – what is it. [Електронний ресурс] – Режим доступу: http://www.daniweb.com/software-development/java/threads/85391/from-beginner-in-java-atomic-type-what-is-it (дата звернення: 26.03.2013). – Назва з екрану.
3. Interlocked класс — Режим доступу: http://msdn.microsoft.com/ru-ru/library/5kczs5b5.aspx (дата звернення 26.03.13). - Назва з екрану.
4. Java Concurrency - Atomic-Variables [Електронний ресурс] – Режим доступу: http://www.baptiste-wicht.com/2010/09/java-concurrency-atomic-variables/ (дата звернення:26.03.13). – Назва з екрану.
5. Package java.util.concurrent.atomic [Електронний ресурс] – Режим доступу: http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/atomic/package-summary.html (дата звернення: 26.03.13). – Назва з екрану.
6. Антонов А.С. Параллельное программирование с использованием технологии OpenMP: Учебное пособие.-М.: Изд-во МГУ, 2009. - 77 с.
7. Выбор примитивов синхронизации для минимизации издержек [Електронний ресурс] – Режим доступу: http://software.intel.com/ru-ru/articles/choosing-appropriate-synchronization-primitives-to-minimize-overhead (дата звернення 26.03.13). – Назва з екрану.
8. Многозадачность[Електронний ресурс] – Режим доступу: http://www.ada-ru.org/V-0.4w/part_1/ch_15.html(дата звернення 26.03.2013). – Назва з екрану.
9. Параллельные заметки №5 — продолжаем знакомиться с Open MP [Електронний ресурс] – Режим доступу: http://habrahabr.ru/company/intel/blog/88574/ (дата звернення:26.03.13). – Назва з екрану.

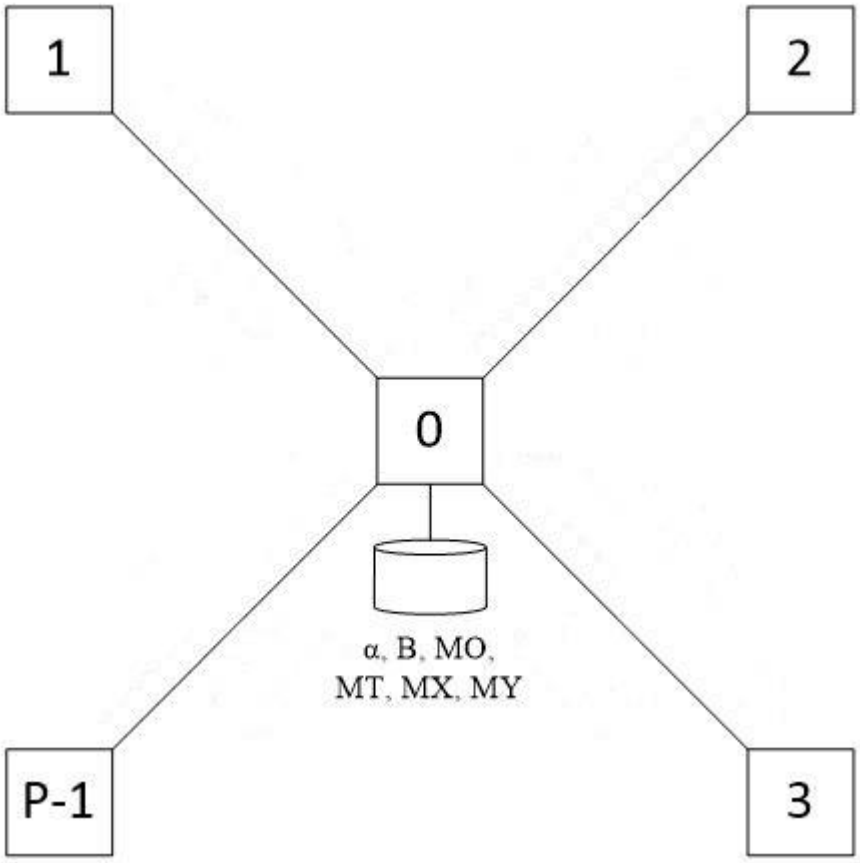
10. Сравнение C.Sharp и Java– Режим доступу: http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_C_Sharp_%D0%B8_Java(дата звернення 26.03.2013). – Назва з екрану.

РОЗДІЛ 6. Додатки

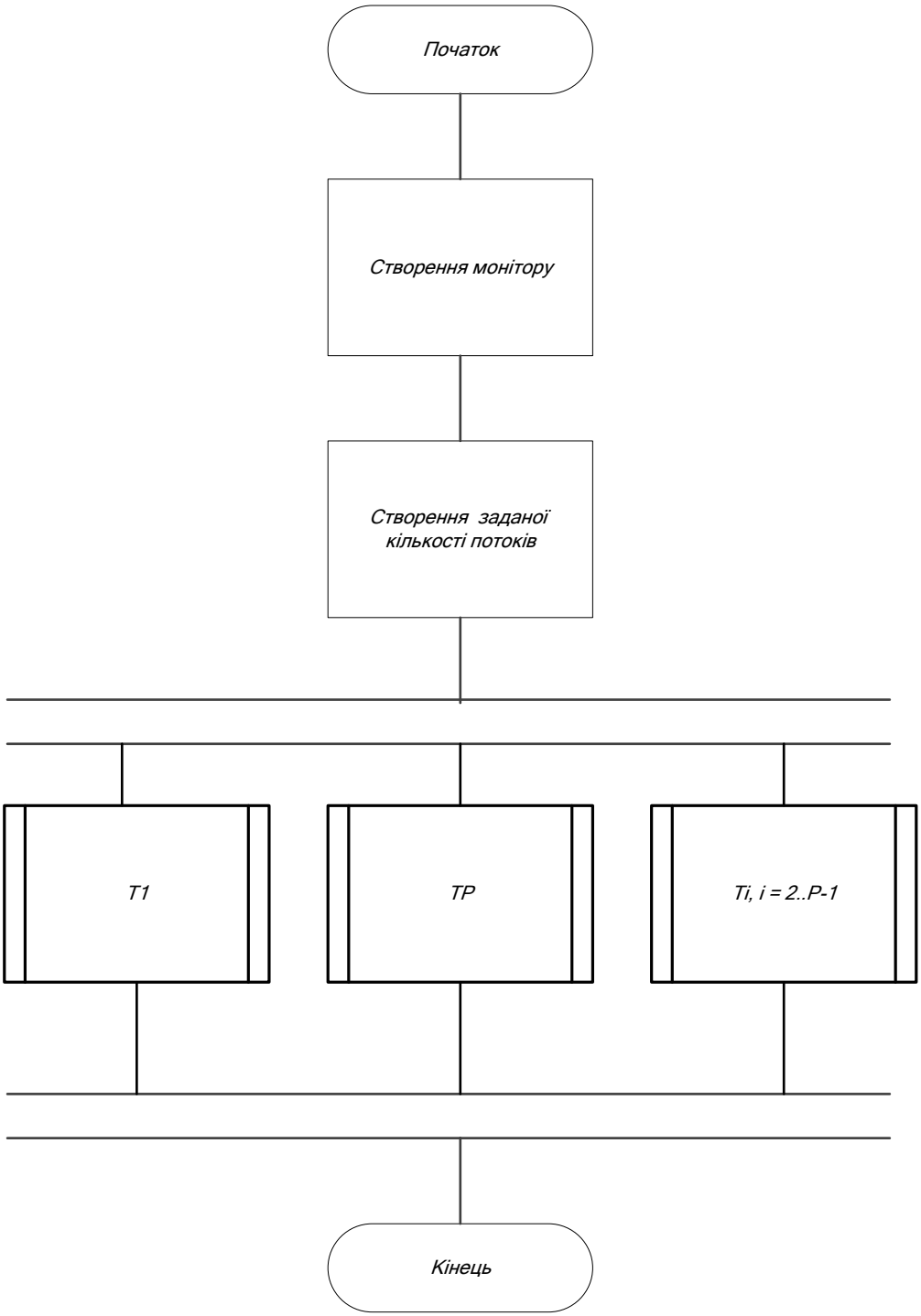
Додаток А. Структурна схема ПКС з ЗП
Додаток Б. Структурна схема ПКС з ЛП
Додаток В. Алгоритм основної програми для ПКС з ЗП
Додаток Г. Алгоритм процесів в програмі для ПКС з ЗП
Додаток Д. Алгоритм основної програми для ПКС з ЛП
Додаток Е. Алгоритм процесів у програмі для ПКС з ЛП
Додаток Ж. Лістінг програми для ПКС з ЗП
Додаток З. Лістінг програми для ПКС з ЛП.



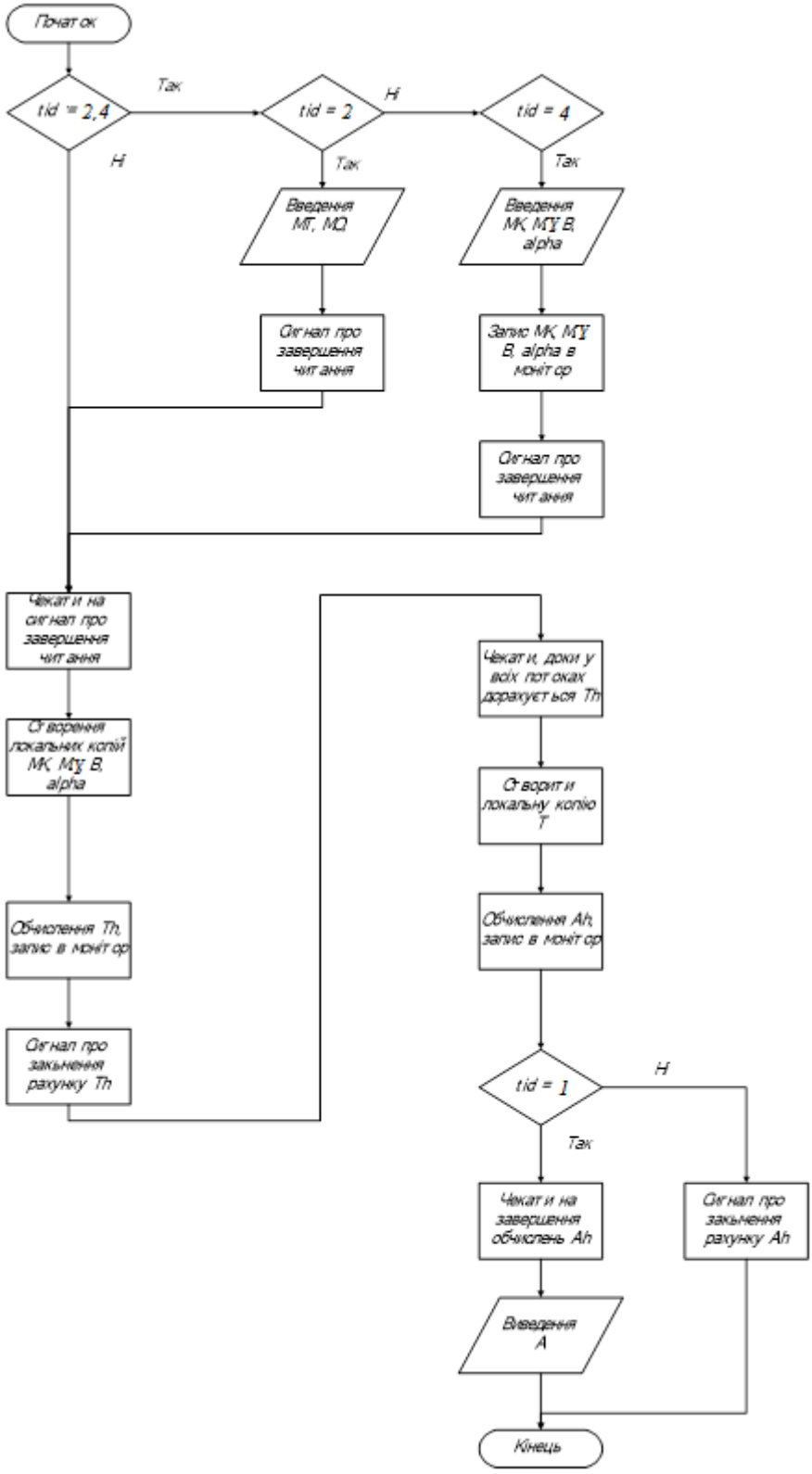
					ІАЛЦ 462637.001		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Наумова Х.С..			*Додаток А. Структурна схема ПКС з ЗП*	Літ.	Арк. 33 / Акрушів 46
Перевір.		Корочкін О.В..					
Реценз.						33	
Н. Контр.						НТУУ «КПІ»	
Затверд.							



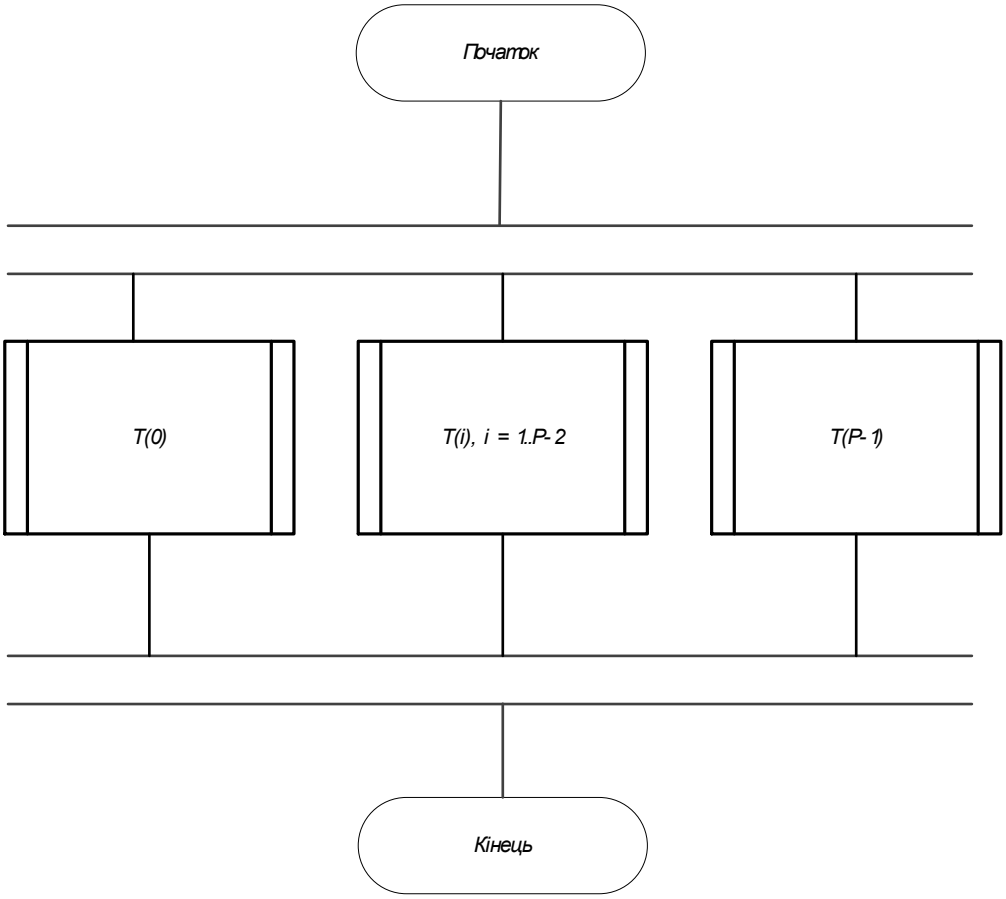
					ІАЛЦ 462637.002		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Наумова Х.С.			Додаток Б. Структурна схема ПКС з ЛП	Літ.	Арк. 1 / Акрушів 1
Перевір.		Корочкін О.В..					
Реценз.						34	
Н. Контр.						НТУУ «КПІ»	
Затверд.							



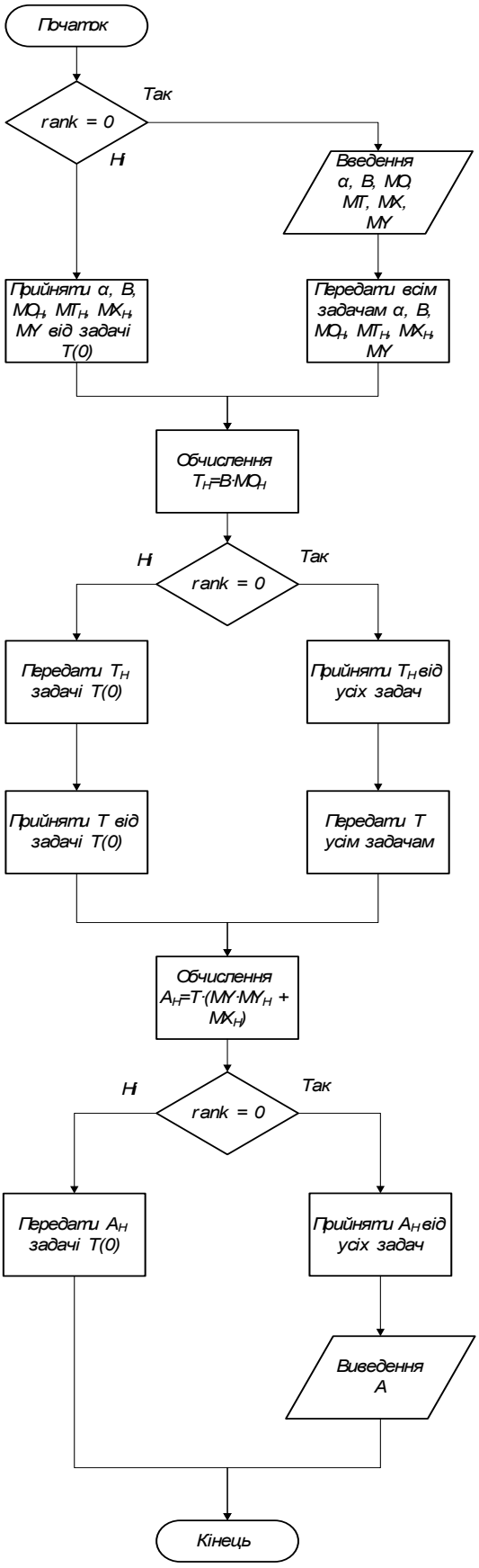
					ІАЛЦ 462637.003		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Наумова Х.С.			Додаток В. Алгоритм програми ПКС з ЗП	Літ.	Арк. 35 / Акрушів 46
Перевір.		Корочкін О.В..					
Реценз.						35	
Н. Контр.						НТУУ «КПІ»	
Затверд.							



					ІАЛЦ 462637.004		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Наумова Х.С..			*Додаток Г. Алгоритм процесу ПКС з ЗП*	Літ.	Арк. / Акрушів
Перевір.		Корочкін О.В..					36 / 46
Реценз.							
Н. Контр.						НТУУ «КПІ»	
Затверд.							



					ІАЛЦ 462637.005		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Наумова Х.С.			*Додаток Д. Алгоритм програми ПКС з ЛП*	Літ.	Арк. 1 / Акрушів 1
Перевір.		Корочкін О.В..					
Реценз.						37 НТУУ «КПІ»	
Н. Контр.							
Затверд.							



					ІАЛЦ 462637.006	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Наумова Х.С.			*Додаток Е. Алгоритм процесів ПКС з ЛП*	Літ. / Арк. 1 / Акрушів 1
Перевір.		Корочкін О.В..				
Реценз.						38
Н. Контр.						НТУУ «КПІ»
Затверд.						

Додаток Ж. Лістінг програми ПКС ЗП

```
//PRO course work Java
//Naumova Kristina IO-01
//A = (B*MO)*(MY*MT+alpha*MX)
//22.03.2013

public class Runner {
	public static void main(String[] args) throws InterruptedException {
		int P = 6;
		int N = 120;
		Monitor mntr = new Monitor(P);
		long startTime = System.currentTimeMillis();
		Task finalTask = null;
		for (int i = 0; i < P; i++) {
			finalTask = new Task(i + 1, N, P, mntr);
		}
		finalTask.join();
		long endTime = System.currentTimeMillis();
		System.out.println("Time elapsed: " + (endTime - startTime) + "
ms");
	}
}

public class Monitor {
	private int p;

	private int F1 = 0;
	private int F2 = 0;
	private int F3 = 0;

	private int[][] MY;
	private int[] B;
	private int alpha;
	private int[] T;

	public Monitor(int n) {
		super();
		p = n;
	}

	public synchronized void writeData(int a, int[] b, int[][] my) {
		MY = new int[my.length][my[0].length];

		alpha = a;

		for (int i = 0; i < my.length; i++) {
			for (int j = 0; j < my[0].length; j++) {
				MY[i][j] = my[i][j];
			}
		}

		B = new int[b.length];
		for (int i = 0; i < b.length; i++) {
			B[i] = b[i];
		}

	}

	public synchronized void writeT(int[] t, int from, int to) {

		T = new int[t.length];
		for (int i = from; i < to; i++) {
			T[i] = t[i];
		}
```

```
}


public synchronized int[][] copyMY() {
	int[][] res = new int[MY.length][MY[0].length];
	for (int i = 0; i < MY.length; i++) {
		for (int j = 0; j < MY[0].length; j++) {
			res[i][j] = MY[i][j];
		}
	}
	return res;
}

public synchronized int[] copyB() {
	int[] res = new int[B.length];
	for (int i = 0; i < B.length; i++) {
		res[i] = B[i];
	}
	return res;
}

public synchronized int copyAlpha() {
	int res = alpha;
	return res;
}

public synchronized int[] copyT() {
	int[] res = new int[T.length];
	for (int i = 0; i < T.length; i++) {
		res[i] = T[i];
	}
	return res;
}

public synchronized void waitIn() {
	if (F1 < 2) {
		try {
			wait();
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}
}

public synchronized void waitCalc() {
	if (F2 < p - 1) {
		try {
			wait();
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}
}

public synchronized void waitT() {
	if (F3 < p) {
		try {
			wait();
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}
}

public synchronized void signalIn() {
```

```
            F1++;
            if (F1 == 2)
                  notifyAll();
      }

      public synchronized void SignalCalc() {
            F2++;
            if (F2 == p - 1)
                  notify();
      }

      public synchronized void SignalT() {
            F3++;
            if (F3 == p)
                  notifyAll();
      }
}

public class Task extends Thread {
      static int[][] MT;
      static int[][] MX;
      static int[][] MO;
      static int[] A;

      static Monitor mntr;

      static int N;
      int K = 1;
      int Tid;
      int H;
      int P;

      public Task(int id, int N, int p, Monitor m) {
            Tid = id;
            Task.N = N;
            A = new int[N];
            P = p;
            H = N / P;
            mntr = m;
            this.start();
      }

      public void run() {
            System.out.println("Task " + Tid + " started");

            if (Tid == 2) {
                  MT = new int[N][N];
                  MO = new int[N][N];
                  for (int i = 0; i < N; i++) {
                        for (int j = 0; j < N; j++) {
                              MO[i][j] = K;
                              MT[i][j] = K;
                        }
                  }

                  mntr.signalIn();
            }
            if (Tid == 4) {
                  int[] B = new int[N];
                  int[][] MY = new int[N][N];
                  int alpha = 1;
                  for (int i = 0; i < N; i++) {
                        B[i] = K;
                        for (int j = 0; j < N; j++) {
                              MY[i][j] = K;
```

```
                }
        }
        MX = new int[N][N];
        for (int i = 0; i < N; i++) {
                for (int j = 0; j < N; j++) {
                        MX[i][j] = K;
                }
        }
        mntr.writeData(alpha, B, MY);
        mntr.signalIn();
}

mntr.waitIn();

int[][] MYcopy;
MYcopy = mntr.copyMY();

int[] Bcopy;
Bcopy = mntr.copyB();

int alphaCopy = mntr.copyAlpha();


int[] tTemp = new int[N];

int from = (Tid - 1) * H;
int to = Tid * H;

for (int i = from; i < to; i++) {
        tTemp[i] = 0;
        for (int j = 0; j < N; j++) {
                tTemp[i] += Bcopy[j] * MO[i][j];
        }
}
mntr.writeT(tTemp, from, to);
mntr.SignalT();
mntr.waitT();
int[] Tcopy;
Tcopy = mntr.copyT();

int[][] temp1 = new int[N][N];
int[][] temp2 = new int[N][N];

for (int i = from; i < to; i++) {
        A[i] = 0;
        for (int j = 0; j < N; j++) {
                temp1[i][j] = 0;
                temp2[i][j] = 0;
                for (int k = 0; k < N; k++) {
                        temp1[i][j] += MT[i][k] * MYcopy[k][j];
                }
                A[i] += Tcopy[j] * (temp1[i][j] + alphaCopy *
                MX[i][j]);
        }
}
if (Tid != 1) {
        mntr.SignalCalc();
        System.out.println("Task " + Tid + " finished");
} else {
        mntr.waitCalc();
        if (N < 20) {
                System.out.println();
                for (int i = 0; i < N; i++)
                        System.out.print(A[i] + " ");
                System.out.println();
```

```
                    }
                    System.out.println("Task " + Tid + " finished");

                }
            }
        }
```

Додаток 3. Лістінг програми ПКС ЛП

```
// PRO course work MPI
// Naumova Kristina IO-01
// A = (B*MO) * (MY*MT + alpha*MX)
// 03.05.2013


#include <mpi.h>
#include "data.h"

int N = 6,
	P,
	H;

int main(int args, char* argv[])
{
	MPI_Init(&args, &argv);
	int rank;
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &P);
	H = N / P;

	cout << "Task " << rank + 1 << " started" << endl;

	int alpha = 0, rows_MO_MT_MX = H;
	if(rank == 0)
	{
		rows_MO_MT_MX = N;
	}

	Vector B(N);
	Matrix MO(rows_MO_MT_MX, N);
	Matrix MY(N);
	Matrix MX(rows_MO_MT_MX, N);
	Matrix MT(rows_MO_MT_MX, N);


	double start_time, elapsed_time;
	// 1. Введення даних
	if(rank == 0)
	{
		start_time = MPI_Wtime();
		alpha = 1;
		B.fill(1);
		MO.fill(1);
		MT.fill(1);
		MX.fill(1);
		MY.fill(1);
		/*alpha = 2;
		for(int i = 0; i < B.cols; i++)
		{
			B.set(i, i);
		}
		for(int i = 0; i < MO.cols; i++)
		{
			for(int j = 0; j < MO.rows; j++)
			{
```

```
                        MO.set(j, i, 10+j*10+i);
                        MT.set(j, i, 100+j*10+i);
                        MX.set(j, i, 200+j*10+i);
                        MY.set(j, i, 300+j*10+i);
                    }
                }*/
        }


        // 2. Передати число alpha всім задачам
        MPI_Bcast(&alpha, 1, MPI_INT, 0, MPI_COMM_WORLD);

        // 3. Передати вектор B всім задачам
        MPI_Bcast(B.get_adress(0), N, MPI_INT, 0, MPI_COMM_WORLD);

        // 4. Передати H стовпців матриці MO всім задачам
        if(rank == 0)
        {
                MO.transpose();
                MPI_Scatter(MO.get_adress(0), H*N, MPI_INT, MO.get_adress(0), H*N,
MPI_INT, 0, MPI_COMM_WORLD);
        }
        else
        {
                MPI_Scatter(NULL, 0, MPI_DATATYPE_NULL, MO.get_adress(0), H*N, MPI_INT,
0, MPI_COMM_WORLD);
        }

        // 5. Передати матрицю MY всім задачам
        MPI_Bcast(MY.get_adress(0), N*N, MPI_INT, 0, MPI_COMM_WORLD);


        // 6. Передати H стовпців матриць MT, MX всім задачам
        if(rank == 0)
        {
                MT.transpose();
                MX.transpose();
                MPI_Scatter(MX.get_adress(0), H*N, MPI_INT, MX.get_adress(0), H*N,
MPI_INT, 0, MPI_COMM_WORLD);
                MPI_Scatter(MT.get_adress(0), H*N, MPI_INT, MT.get_adress(0), H*N,
MPI_INT, 0, MPI_COMM_WORLD);
        }
        else
        {
                MPI_Scatter(NULL, 0, MPI_DATATYPE_NULL, MX.get_adress(0), H*N, MPI_INT,
0, MPI_COMM_WORLD);
                MPI_Scatter(NULL, 0, MPI_DATATYPE_NULL, MT.get_adress(0), H*N, MPI_INT,
0, MPI_COMM_WORLD);
        }

        // 7. Обчислення TH = B * MOH
        Vector T_H(H);
        vector_matrix_multiply(0, H, B, MO, T_H);

        // 8. Зібрати вектор T в першій задачі
        Vector T(N);
        if(rank == 0)
        {
```

```
		MPI_Gather(T_H.get_adress(0), H, MPI_INT, T.get_adress(0), H, MPI_INT,
0, MPI_COMM_WORLD);
	}
	else
	{
		MPI_Gather(T_H.get_adress(0), H, MPI_INT, NULL, 0, MPI_INT, 0,
MPI_COMM_WORLD);
	}

	// 9. Розіслати вектор T усім задачам
	MPI_Bcast(T.get_adress(0), N, MPI_INT, 0,  MPI_COMM_WORLD);


	// 10. Обчислення AH = T*(MY*MTH + alpha*MXH)
	Vector A_H(H);
	calculation(0, H, alpha, T, MT, MX, MY, A_H);

	// 11. Зібрати вектор A в першій задачі
	Vector A(N);
	if(rank == 0)
	{
		MPI_Gather(A_H.get_adress(0), H, MPI_INT, A.get_adress(0), H, MPI_INT,
0, MPI_COMM_WORLD);
	}
	else
	{
		MPI_Gather(A_H.get_adress(0), H, MPI_INT, NULL, 0, MPI_INT, 0,
MPI_COMM_WORLD);
	}

	// 12. Виведення результату
	if(rank == 0)
	{
		elapsed_time = MPI_Wtime() - start_time;
		if(N < 15)
		{
			A.output();
		}
		cout << endl << "Time elapsed: " << (int)(elapsed_time * 1000) << " ms"
<< endl;
	}

	cout << "Task " << rank + 1 << " finished" << endl;

	MPI_Finalize();

	if(rank == 0)
	{
		system("pause");
	}
	return 0;
}

#include "data.h"

void vector_matrix_multiply(const int start, const int end,
	const Vector &B, const Matrix &MO, Vector &T)
{
```

```
	int sum;
    for (int i = start; i < end; i++)
	{
		sum = 0;
		for (int j = 0; j < B.cols; j++)
		{
			sum += B.get(j) * MO.get(i, j);
		}
		T.set(i, sum);
    }
}

//AH = T(MY*MTH + alpha*MXH)
void calculation(const int start, const int end, const int alpha,
	const Vector &T, const Matrix &MT, const Matrix &MX, const Matrix &MY,
	Vector &A)
{
	long int sum1, sum2;

	for(int i = start; i < end; i++) {
		sum2 = 0;
		for(int j = 0; j < MY.cols; j++) {
			sum1 = 0;
			for(int k = 0; k < MY.cols; k++) {
				sum1 += MT.get(i, k) * MY.get(j, k);
			}
			sum1 += alpha * MX.get(i, j);
			sum2 += T.get(j) * sum1;
		}
		A.set(i, sum2);
	}
}

#include "matrix.h"


Matrix::Matrix(int rows, int cols) :
	Vector(rows * cols),
	rows(rows),
	cols(cols)
{
}


Matrix::Matrix(int N) :
	Vector(N * N),
	rows(N),
	cols(N)
{
}

Matrix::Matrix(const Matrix &other) :
	Vector(other.rows * other.cols),
	rows(other.rows),
	cols(other.cols)
{
	for(int i = 0; i < this->rows; i++)
	{
```

```
			for(int j = 0; j < this->cols; j++)
			{
				Matrix::set(i, j, other.get(i, j));
			}
		}
}


Matrix::~Matrix()
{
}


void Matrix::output()
{
	for(int i = 0; i < this->rows; i++)
	{
		for(int j = 0; j < this->cols; j++)
		{
			cout << Vector::get(i * this->cols + j) << "\t";
		}
		cout << endl;
	}
}

void Matrix::get_column(int col, Vector &vector)
{
	assert(col < this->cols);
	for(int i = 0; i < this->rows; i++)
	{
		Vector::set(i, Vector::get(i * this->cols + col));
	}
}

void Matrix::transpose()
{
	Matrix copy(*this);
	for(int i = 0; i < this->rows; i++)
	{
		for(int j = 0; j < this->cols; j++)
		{
			Matrix::set(j, i, copy.get(i, j));
		}
	}
}



#include "Vector.h"


Vector::Vector(int cols) :
	cols(cols),
	data(new long int[cols])
{
}
```

```
Vector::~Vector()
{
	delete [] data;
}

void Vector::fill(long int value)
{
	for(int i = 0; i < this->cols; i++)
	{
		set(i, value);
	}
}

void Vector::output()
{
	for(int i = 0; i < this->cols; i++)
	{
		cout << this->data[i] << " ";
	}
}
```

```
#pragma once

#include <assert.h>
#include <iostream>
using namespace std;

class Vector
{
public:
	Vector(int cols);
	~Vector();

	void * get_adress(int element)
	{
		return this->data + element;
	}

	long int get(int i) const
	{
		assert(i < this->cols);
		return this->data[i];
	}

	void set(int i, long int value)
	{
		assert(i < this->cols);
		this->data[i] = value;
	}

	void fill(long int value);
	void output();

	const int cols;

private:
	long int * data;
};
```

```
#pragma once
#include "vector.h"

class Matrix :
	public Vector
{
public:
	Matrix(int rows, int cols);
	Matrix(int N);
	Matrix(const Matrix &other);
	~Matrix();

	long int get(int i, int j) const
	{
		assert(i < this->rows);
		assert(j < this->cols);
		return Vector::get(i * cols + j);
	}

	void set(int i, int j, long int value)
	{
		assert(i < this->rows);
		assert(j < this->cols);
		Vector::set(i * cols + j, value);
	}

	void output();
	void get_column(int col, Vector &vector);
	void transpose();


	const int rows;
	const int cols;


};

#pragma once

#include "vector.h"
#include "matrix.h"

void vector_matrix_multiply(const int start, const int end,
	const Vector &B, const Matrix &MO, Vector &T);

void calculation(const int start, const int end, const int alpha,
	const Vector &T, const Matrix &MT, const Matrix &MX, const Matrix &MY,
	Vector &A);
```