НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. ІГОРЯ СІКОРСЬКОГО»

Кафедра обчислювальної техніки
(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з дисципліни «Паралельні та розподілені обчислення»
(назва дисципліни)

на тему: «Порівняння реалізації механізму семафорів секції в мовах і бібліотеках паралельного програмування»

Студента 3 курсу ІО-43 групи
напряму підготовки 6.123 «Комп'ютерна інженерія»
Крута В. В.
(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 – 2017 рік

Національний технічний університет України
"Київський політехнічний інститут ім. Ігоря Сікорського"

Факультет (інститут) інформатики та обчислювальної техніки
(повна назва)

Кафедра обчислювальної техніки
(повна назва)

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.123 «Комп'ютерна інженерія»
(шифр і назва)

З А В Д А Н Н Я

НА КУРСОВУ РОБОТУ СТУДЕНТУ

Круту Владиславу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка програмного забезпечення для паралельних комп'ютерних систем»
керівник роботи Корочкін Олександр Володимирович к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи 12 березня 2017 р.

3. Вхідні дані до роботи
- порівняння реалізації механізму семафорів секції в мовах і бібліотеках паралельного програмування
- математична задача
- структури ПКС ОП та ПКС ЛП
- мови і бібліотеки програмування: Ада, Java, C#, WinAPI
- засоби організації взаємодії процесів: захищений модуль, механізм рандеву мови Ада

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- порівняння реалізації механізму семафорів секції 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.

7. Дата видачі завдання 28.02.17

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання КР	Строк виконання етапів КР
1	Виконання розділу 1	**13.03.2017**
2	Виконання розділу 2	**03.04.2017**
3	Виконання розділу 3	**24.04.2017**
4	Оформлення КР	**8.05.2017**
5	Перевірка КР викладачем	**11.05.2017**
6	Захист КР	**18.05.2017**

Студент _____ Крут В. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Корочкін О. В.
(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

В даній курсовій роботі в першому розділі оглянуто різноманітні реалізації семафорів – спеціальних механізмів для вирішення задачі взаємного виключення та задачі синхронізації. Описані реалізації в мовах Ада, Java, C# та у бібліотеці Win32. Порівняно їх недоліки та переваги.

Наведено список використаних скорочень та список використаної літератури.

РОЗДІЛ 1. РЕАЛІЗАЦІЯ СЕМАФОРІВ

Семафор – це універсальний механізм для організації взаємодії процесів[8]. Семафор вирішує задачу взаємного виключення та задачу синхронізації. Зазвичай семафор реалізовано спеціальним захищеним типом. Захищеність означає, що заборонено усі операції над семафорами, окрім трьох – створення семафору, P(S) та V(S), де S – семафор. P(S) – реалізація входу в критичну ділянку у задачі взаємного виключення або очікування сигналу в задачі синхронізації[1]. В цій операції перевіряється значення семафору, якщо S=0, то процес блокується, інакше виконує дію S=S-1 та входить у критичну ділянку або продовжує виконання. V(S)- реалізація виходу з критичної ділянки у задачі взаємного виключення або відправлення сигналу в задачі синхронізації. Ця операція виконує дію S=S+1, що є виходом з критичної ділянки або відправкою сигналу[11]. Семафори реалізовано у мовах Ада, Java, C#, у бібліотеці Win32[1].

1.1 Реалізація семафорів у мові Ада

У мові Ада для семафорів створено спеціальний тип Suspension_object, який є лімітованим захищеним типом. Для використання цього типу необхідно підключити пакет Ada.Synchronous_Task_Control[7].

У цьому пакеті визначені такі операції:

- procedure Suspend_until_true (S: in out Suspension_object), що є реалізацією операції P(S);

- procedure Set_true (S: in out Suspension_object), що є реалізацією операції V(S);

- procedure Set_false (S: in out Suspension_object), що є додатковою операцією та встановлює семафору значення false;

- function Current_state (S: Suspension_object) return Boolean, що є додатковою операцією та перевіряє значення семафору[6].

Семафор у мові Ада може приймати два значення – true та false. При створенні семафору йому за замовчанням присвоюється значення false[12].

1.2 Реалізація семафорів у мові Java

В мові Java семафори реалізовано класом Semaphore пакету java.util.concurent.*. Цей клас має два конструктори – Semaphore (int X), де X – початкове значення семафору та Semaphore (Boolean X), де X – поведінка P(S) та V(S), якщо X=true, то по черзі, якщо X=false, то випадково[2]. В класі Semaphore визначено такі операції для роботи з семафорами:

- acquire() – метод, що реалізує операцію P(S), підчас якої виконується перевірка значення дозволів семафору, якщо значення семафору більше або дорівнює одиниці, то від значення семафору віднімається одиниця і потік продовжує своє виконання, якщо значення менше, то потік блокується до моменту коли умова виконується. Метод acquire() має модифікацію в якій задається число, на скільки зменшити семафор. Також в Java реалізований схожий метод acquireUninterruptibly[5], цей метод отримує дозвіл від семафору, блокуючи потік, доки дозвіл не буде доступний. Якщо є доступний дозвіл, то отримує його и миттєво повертається, зменшуючи кількість доступних дозволів на одиницю. Якщо немає доступних дозволів, то даний потік стає недіючим для розкладу потоків і очікує, доки інший потік не викличе метод release() для цього семафору і даний потік стає наступним, кому буде надано дозвіл. Якщо даний потік переривається, очікуючи на дозвіл, то він продовжить очікувати, але час, за який потік отримає дозвіл, буде змінено порівняно з часом, за який він отримав би дозвіл без переривання. Коли потік повертається з цього методу, його статус переривання буде встановлено. Тобто цей метод схожий до методу acquire(), але переривання не розблокує потік, а лише змінить час його очікування на дозвіл від семафору. У даного методу є модифікація, за якої задається кількість дозволів.
- availablePermits() – метод, що повертає кількість наявних в цьому семафорі дозволів.
- release() – метод, що реалізує операцію V(S), підчас якої виконується збільшення кількості дозволів (повернення) в семафор на один. Якщо всі потоки намагаються отримати дозвіл, то вибирається потік і йому надається дозвіл. Цей потік

включається в список планування потоків. Даний метод може приймати, в якості параметру, на скільки дозволів збільшити семафор.

- drainPermits() – метод, що отримує та повертає всі дозволи, котрі зразу стають доступними.
- getQueuedThreads() - повертає колекцію потоків, які можуть очікувати дозвіл. Оскільки фактичний набір потоків може змінитися під час виконання методу, результат може бути не достовірним. Елементи колекції не мають порядку.
- getQueueLength() - повертає оцінку числа потоків, які очікують дозволу. Значення тільки оцінка, тому що число потоків може мінятися динамічно під час виконання методу. Цей метод призначений для використання в сфері моніторингу стану системи, а не для синхронізації.
- hasQueuedThreads() - надсилає запит, чи чекають якісь потоки методу acquire().
- isFair() - повертає істину, якщо справедливість семафору встановлена в істину.
- reducePermits() - зменшує кількість дозволів. Цей метод може бути корисним в підкласах, котрі використовують семафори для моніторингу ресурсів. Основна відмінність від методу acquire() в тому, що він не блокує потік доки дозвіл не стане доступним. Даний метод може приймати, в якості параметру, на скільки зменшити кількість дозволів, якщо без параметрів то на 1.
- tryAcquire() - отримує вказану кількість дозволів від цього семафора, тільки якщо всі вони доступні під час виклику методу і відразу ж повертається зі значенням доступних дозволів скороченим на задану величину. Якщо дозволів немає, то цей метод повертає негайно зі значенням помилки і число доступних дозволів залишається незмінним.
- В мові Java на відміну від мови Ада лічильники семафорів є змінними не булевського типу, а чисельного типу integer. Тобто семафори можуть бути не тільки двійковими, а й багатозначними.

1.3 Реалізація семафорів в мові C#

Семафор в C# має декілька варіацій: Semaphore, SemaphoreSlim[3].

Semaphore обмежує кількість потоків, які можуть отримати доступ до ресурсу або пулу ресурсів одночасно. Даний семафор має декілька конструкторів, а саме з значеннями: максимальної кількості одночасних записів і можливо резервування деяких записів; максимальної кількості одночасних записів і можливо резервування деяких записів для виклику потоку, і можливо, із зазначенням найменування об'єкта системного семафора; максимальної кількості одночасних записів і можливо резервування деяких записів для виклику потоку, і можливо, із зазначенням найменування об'єкта системного семафора, і вказати змінну, яка отримує значення, яке вказує, чи є новий семафор системи новостворений та визначення контролю безпеки доступу до системи семафора. В класі Semaphore визначені наступні методи:

- WaitOne() - блокує даний потік до отримання сигналу. Цей метод має декілька модифікацій з різними параметрами за яких до поведінки методу додаються такі дії: спостерігання за CancellationToken; використовуючи 32-розрядне число, яке визначає тайм-аут; використовуючи TimeSpan щоб вказати тайм-аут; використовуючи 32-розрядне число, яке визначає очікування, і вказівки, чи потрібно виходити з домену синхронізації до закінчення очікування; використовуючи TimeSpan, який визначає час очікування, і вказівки, чи потрібно виходити з домену синхронізації до закінчення очікування;
- Dispose() - звільняє всі ресурси, використовувані поточним екземпляром класу WaitHandle.
- Release()- виходить з семафора і повертає попереднє значення лічильника. Є модифікація для виходу з семафору задану кількість разів.
- TryOpenExisting() - відкриває вказаний іменований семафор, якщо він вже існує, і повертає значення, яке вказує, чи є операція успішної. Даний метод має модифікацію в якій: метод відкриває вказаний іменований семафор, якщо він вже існує, з бажаною безпекою доступу і повертає значення, яке вказує, чи є операція успішною[5].

SemaphoreSlim легка альтернатива семафора, який обмежує кількість потоків, які можуть отримати доступ до ресурсу або пулу ресурсів одночасно. Даний семафор має два конструктори: конструктор ініціалізації нового екземпляра класу SemaphoreSlim, з зазначенням початкового кількість запитів, які можуть бути надані одночасно; та конструктор ініціалізації нового екземпляра класу SemaphoreSlim, з зазначенням початкового та максимального числа запитів, які можуть бути надані одночасно. SemaphoreSlim має методи:

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

Обидва семафори в C# можуть викликати виключення SemaphoreFullException - виключення, яке викидається при Semaphore.Release, коли кількість вже на максимумі[13].

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

1.4 Реалізація семафорів у бібліотеці Win32

В бібліотеці Win32 семафори є змінними спеціального типу HANDLE, за якими слідкує сама система. В цій бібліотеці лічильники семафорів також є багатозначними. Для семафорів визначені такі операції:

- HANDLE CreateSemaphore (LPSECURITY_ATTRIBUTES lpSemaphoreAttributes, LONG lInitialCount, LONG lMaximumCount, LPCTSTR lpName) – операція, що створює семафор. lpSemaphoreAttributes – параметри захисту, зазвичай встановлено значення NULL. lInitialCount – початкове значення, воно має бути більше за 0 або рівним 0, та меншим за lMaximumCount або рівним йому. Стан семафору сигнальний, коли це число більше за 0 та не сигнальний, коли рівне 0. Значення семафору зменшується на 1 коли функція очікування розблоковує потік, який чекав на цей семафор. Значення семафору збільшується на визначене значення викликанням функції ReleaseSemaphore. lMaximumCount – максимальне значення, має бути більше за 0. lpName – ім’я семафору, яке має бути не більше за MAX_PATH та може містити в собі будь-які символи, окрім зворотного слешу(\). Порівняння імен регістрозалежне. Якщо lpName співпадає з вже існуючим семафором, то lInitialCount та lMaximumCount ігноруються, тому що вони вже були встановлені при створенні. Якщо lpName є NULL, то семафор створюється без імені. Якщо функція успішна, то вона повертає семафор типу HANDLE. Якщо іменований об’єкт семафору існував до виклику функції, то функція GetLastError повертає ERROR_ALREADY_EXISTS. В інших випадках GetLastError повертає нуль. Якщо CreateSemaphore є невдалою, то повертає NULL. Для поширених відомостей про помилку потрібно викликати функцію GetLastError.

- HANDLE WaitForSingleObject (HANDLE hHandle, DWORD dwMilliseconds) – реалізація операції P(S), hHandle – семафор, dwMilliseconds – визначає час тайм-ауту в мілісекундах. Функція повертає значення коли інтервал часу скінчується, навіть якщо стан семафору несигнальний. Якщо dwMilliseconds дорівнює нулю, то функція визначає стан семафору та повертається миттєво. Якщо dwMilliseconds дорівнює INFINITE, тайм-аут функції ніколи не наступає. Якщо виконання функції успішне, то значення, що вона повертає, визначає яка подія

змусила її повернутися. Є два такі значення: WAIT_OBJECT_0, яке повертається коли стан семафору стає сигнальним та WAIT_TIMEOUT, яке повертається коли трапляється тайм-аут та стан семафору є несигнальним. WAIT_FAILED вказує на не успішне виконання функції. Очікування на некоректний семафор змушує функцію повернути WAIT_FAILED. Для поширених відомостей про помилку потрібно викликати функцію GetLastError.

- BOOL ReleaseSemaphore (HANDLE hSemaphore, LONG lReleaseCount, LPLONG lpPreviousCount) – реалізація операції V(S), hSemaphore – семафор, який повертається функцією CreateSemaphore. lReleaseCount – число, на яке його потрібно збільшити, це число повинне бути більше за нуль. Якщо додавання цього числа до значення, яке вже є, зробить його більше за максимальне значення семафору, то додавання не відбудеться і функція поверне FALSE. lpPreviousCount – попереднє значення семафору. Може бути NULL, якщо попереднє значення не потребується. Якщо функція успішна, то значення, яке вона повертає, є ненульовим. Якщо ні, то повертає нуль. Для поширених відомостей про помилку потрібно викликати функцію GetLastError[3].

1.5 Висновки до розділу 1

1. Розглянуто реалізацію семафорів у різних мовах. Показано, що у будь-яких мовах семафори мають операції створення або конструктори, реалізацію операції P(S) та реалізацію операції V(S), що є основними операціями для використання семафорів.

2. Розглянуто сутність семафорів у різних мовах. Було виявлено, що в мові Ада семафори можуть бути тільки двійковими, а в мовах Java, C# та бібліотеці Win32 вони можуть бути багатозначними, що спрощує задачу синхронізації, оскільки не потрібно створювати багато двійкових семафорів для однотипних сигналів.

3. Виявлено, що в реалізації P(S) у мовах Ада та C# та у бібліотеці Win32 може виконуватись тільки одна дія після виконання умови S==1: S=S-1, а у мові Java може виконуватись також S=S-k, що використовується для багатозначних семафорів та зменшує код порівняно з іншими реалізаціями семафору.

4. Виявлено, що в реалізація V(S) у мові Ада значення семафору може збільшитися лише на 1, а у мовах Java, C# та у бібліотеці Win32 на будь-яке число в межах максимального значення семафору.

5. В результаті вивчення документації по реалізаціям семафорів в різних мовах визначено, що в мові Java можна обрати як будуть розблоковуватися процеси – по черзі або випадково. В інших мовах такої можливості немає.

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС ЗІ СП

У даному розділі розроблено програму ПРГ1 для паралельної комп'ютерної системи зі спільною пам'яттю, яка обраховує заданий математичний вираз:

A = sort(d*Z) + S*(MO*MK)

Структурна схема паралельної комп'ютерної системи зі спільною пам'яттю зображена в додатку А.

2.1. Розробка паралельного математичного алгоритму

Паралельний математичний алгоритм для заданого виразу:

1) Rh = sort(Zh);
2) R2h = mergeSort(Rh, Rh);
3) R = mergeSort(R2h, R2h);
4) $A_H = d*R_H + S*(MO_H*MK)$;

Спільний ресурс: d, S, MK.

2.2. Розробка алгоритмів процесів

Дана програма реалізується чотирма потоками й алгоритм виконання процесів показано в таблиці 2.1..

Таблиця 2.1. **Алгоритм роботи потоків**

T1	КД	T2	КД
1) Очікування вводу даних T2-T4; 2) Сортування Rh = sort(Zh); 3) Сигнал про закінчення обрахунку T2-T4; 4) Очікування закінчення обрахунку T2-T4; 5) Сортування R2h = mergeSort(Rh, Rh);	$W_{2,1}$ - $W_{4,3}$ $S_{2,1}$ - $S_{4,3}$ $W_{2,4}$ - $W_{4,6}$	1) Ввід d; 2) Сигнал T1, T3, T4 про ввід даних; 3) Очікування вводу даних T3, T4; 4) Сортування Rh = sort(Zh); 5) Сигнал про закінчення обрахунку T1, T3, T4;	$S_{1,1}$, $S_{3,2}$, $S_{4,3}$ $W_{3,1}$, $W_{4,2}$ $S_{1,4}$, $S_{3,5}$, $S_{4,6}$

Продовження таблиці 2.1.

6) Сигнал про закінчення обрахунку Т2;	$S_{2,4,}$	6) Очікування закінчення обрахунку Т1, Т3, Т4;	$W_{1,3,}$ $W_{3,4,}W_{4,5}$
7) Очікування закінчення обрахунку Т2;	$W_{2,5}$	7) Сортування R2h = mergeSort(Rh, Rh);	
8) Сортування R = mergeSort(R2h, R2h);		8) Сигнал про закінчення обрахунку Т1;	$S_{1,7}$
9) Сигнал про закінчення обрахунку Т2-Т4;	$S_{2,5,}$ $S_{3,6}$, $S_{4,7}$	9) Очікування сигналу про закінчення обрахунку R від Т1;	$W_{1,6}$
10) Копія $d_1 = d$;	КД	10) Копія $d_2 = d$;	КД
11) Копія $S_1 = S$;	КД	11) Копія $S_2 = S$;	КД
12) Копія $МК_1$=МК;	КД	12) Копія $МК_2$=МК;	КД
13) Обрахунок $A_Н = d_1*R_Н + S_1*(МО_Н *МК_1)$;		13) Обрахунок $A_Н = d_2*R_Н + S_2*(МО_Н *МК_2)$;	
14) Сигнал про закінчення обрахунку Т2;	$S_{2,8}$	14) Очікування закінчення обрахунку Т1, Т3, Т4;	$W_{1,7,}W_{3,8,}$ $W_{4,9}$
		15) Вивід А;	

Т3	КД	Т4	КД
1) Ввід Z, МК;		1) Ввід S, МО;	
2) Сигнал Т1, Т2, Т4 про ввід даних;	$S_{1,1,}$ $S_{2,2}$, $S_{4,3}$	2) Сигнал Т1 - Т3про ввід даних;	$S_{1,1,}$ $S_{2,2}$, $S_{3,3}$
3) Очікування вводу даних Т2, Т4;	$W_{2,1,}$ $W_{4,2}$	3) Очікування вводу даних Т2, Т3;	$W_{2,1,}$ $W_{3,2}$
4) Сортування Rh = sort(Zh);		4) Сортування Rh = sort(Zh);	
5) Сигнал про закінчення обрахунку Т1, Т2, Т4;	$S_{1,4,}$ $S_{2,5}$, $S_{4,6}$	5) Сигнал про закінчення обрахунку Т1 - Т3	$S_{1,4,}$ $S_{2,5}$, $S_{3,6}$
6) Очікування сигналу про закінчення обрахунку R від Т1;	$W_{1,3,}$	6) Очікування сигналу про закінчення обрахунку R від Т1;	$W_{1,3}$

Продовження таблиці 2.1.

7) Копія $d_3 = d$;	КД	7) Копія $d_4 = d$;	КД
8) Копія $S_3 = S$;	КД	8) Копія $S_4 = S$;	КД
9) Копія $MK_3=MK$;	КД	9) Копія $MK_4=MK$;	КД
10) Обрахунок $A_H = d_3*R_H + S_3*(MO_H *MK_3)$;		10) Обрахунок $A_H = d_4*R_H + S_4*(MO_H *MK_4)$;	
11) Сигнал про закінчення обрахунку Т2;	$S_{2,7}$	11) Сигнал про закінчення обрахунку Т2;	$S_{2,7}$

2.3. Розробка схеми взаємодії процесів

На основі алгоритму виконання паралельної програми розроблено структурну схему взаємодії задач для вирішення задачі синхронізації та взаємного виключення (Додаток Б).

На схемі зображено потоки Т1 – Т 4, кожен із яких відправляє та отримує сигнали про ввід даних чи закінчення обрахунку за допомогою бар'єрів та подій; спільні ресурси, доступ до яких контролюється функцією lock() та монітором.

2.4. Розробка програми ПРГ 1

Паралельна програма для обрахунку даного математичного виразу зі спільною пам'яттю реалізовано мовою програмування *C#* (додаток В).

Для реалізації паралельного виконання програми використано:

- Клас Program, який виконує потоки паралельно;
- Бар'єри inputBarrier, computeBarrier для вирішення задачі синхронізації;
- Бар'єри firstSortBarrie, twoSortBarrier, які забезпечують паралельне сортування;
- Подію eventSort для завершення сортування;
- Об'єкти objLock, objMonitor для захисту спільних ресурсів;
- Паралельні цикли;
- Метод thread.Start() запускає потоки на виконання;
- Метод thread.Join() блокує основний потік до завершення даного потоку;

2.5. Результати тестування програми ПРГ 1

Тестування виконано на паралельній обчислювальній системі із характеристиками системи:

- процесор Intel Core i5-7100 ядра по 3.9GHz,об’єм кешпам’яті 3 рівня 6Мб;
- операційна система Windows 10 Home;
- середовище розробки Visual Studio 2015;

Для тестування обрано такі розмірності векторів і матриць: N = 800, N = 1600, N = 2400. Програму виконано ПКС із різним значенням кількості ядер P – від 1 до 4.

Результати виміру часу відображаються в таблиці 2.2.

Таблиця 2.2. **Час виконання програми для ПРГ1**

N	T1	T2	T3	T4
800	26.1	13.5	5.1	7.2
1600	226.3	113.7	80.5	62.2
2400	646.2	328	229.9	177.5

На основі даних таблиці 2.2. обчислено значення коефіцієнтів прискорення, де $К_{пр} = T_1/T_P$. Значення наведені в таблиці 2.3.

Таблиця 2.3. **Значення $К_{пр}$ для ПРГ1**

N	P1	P2	P3	P4
900	1	1.89	2.73	3.61
1800	1	1.99	2.81	3.64
2400	1	1.97	2.86	3.68

На основі даних таблиці 2.3. обчислено значення коефіцієнтів ефективності, де $К_{еф} = К_{пр}/P * 100\%$. Значення наведені в таблиці 2.4.

Таблиця 2.4. **Значення $К_{еф}$ для ПРГ1**

N	T1	T2	T3	T4
900	100 %	95 %	91 %	90 %
1800	100 %	100 %	94 %	91 %
2400	100 %	99 %	95 %	92 %

На підставі таблиць 2.3. та 2.4. побудовано графіки поведінки $К_{пр}$ та $К_{еф}$ залежно від N для ПРГ1.

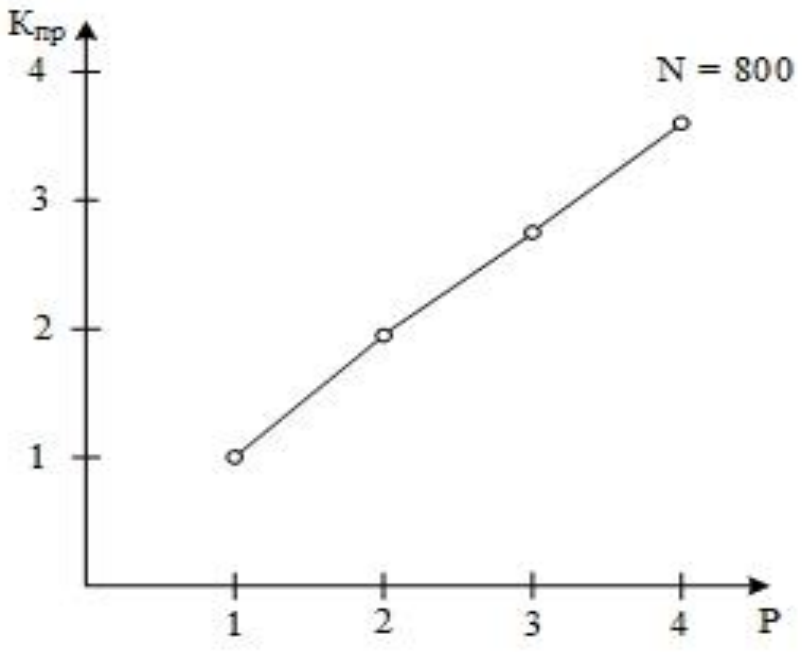

Рис. 2.1. Графік зміни коефіцієнта прискорення $К_{пр}$ залежно від кількості ядер для N = 800

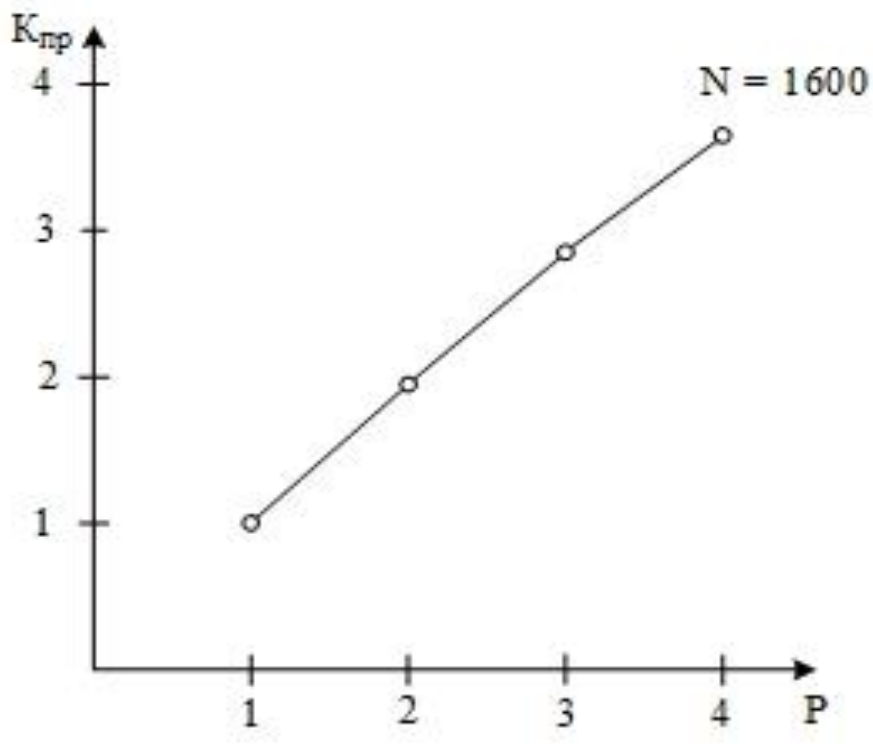

Рис. 2.2. Графік зміни коефіцієнта прискорення $К_{пр}$ залежно від кількості ядер для N = 1600

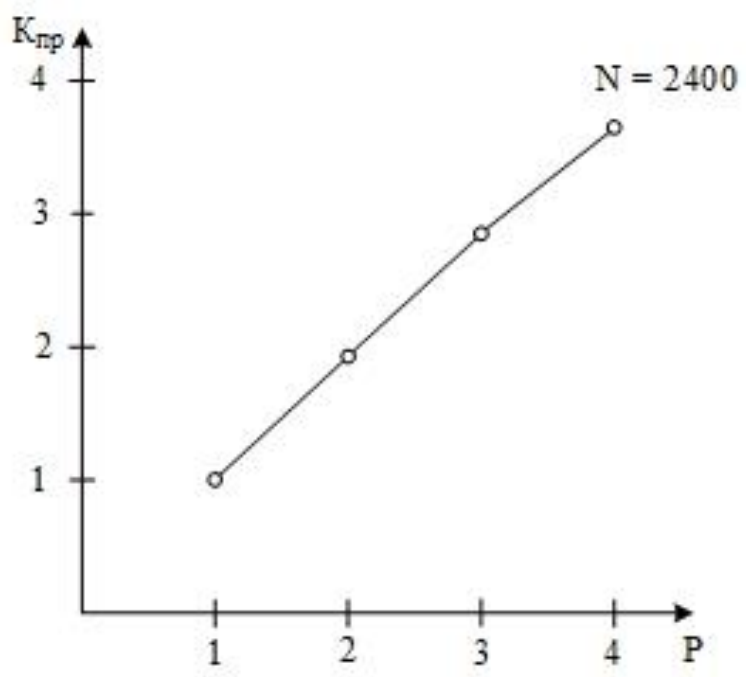

Рис. 2.3. Графік зміни коефіцієнта прискорення $К_{пр}$ залежно від кількості ядер для N = 2400

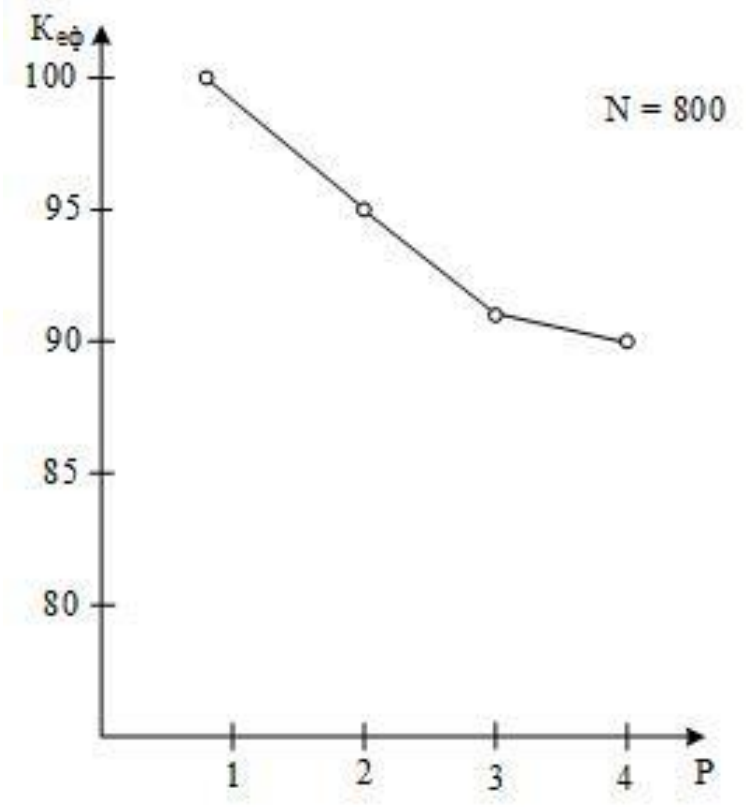

Рис. 2.4. Графік зміни коефіцієнта ефективності $К_{еф}$ залежно від кількості ядер для N = 800

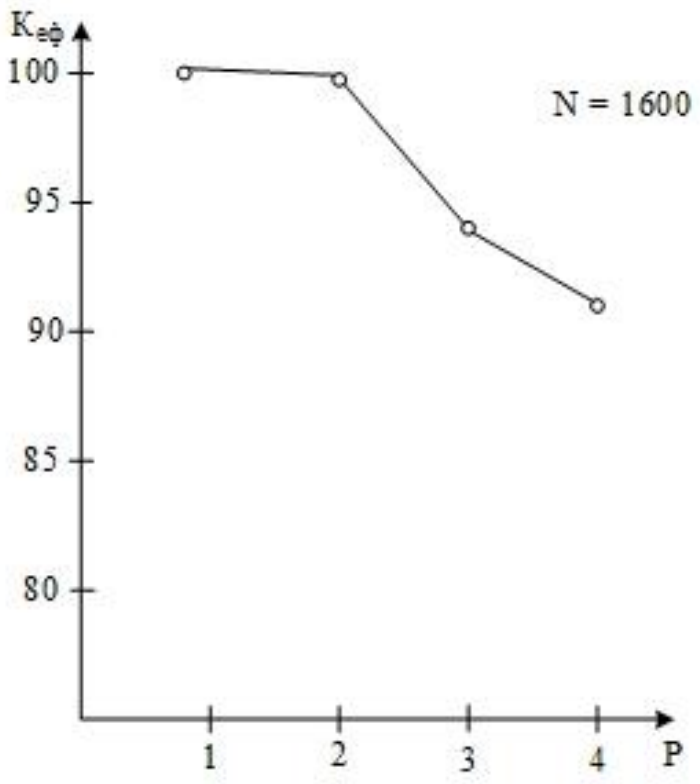

Рис. 2.5. Графік зміни коефіцієнта ефективності $К_{еф}$ залежно від кількості ядер для N = 1600

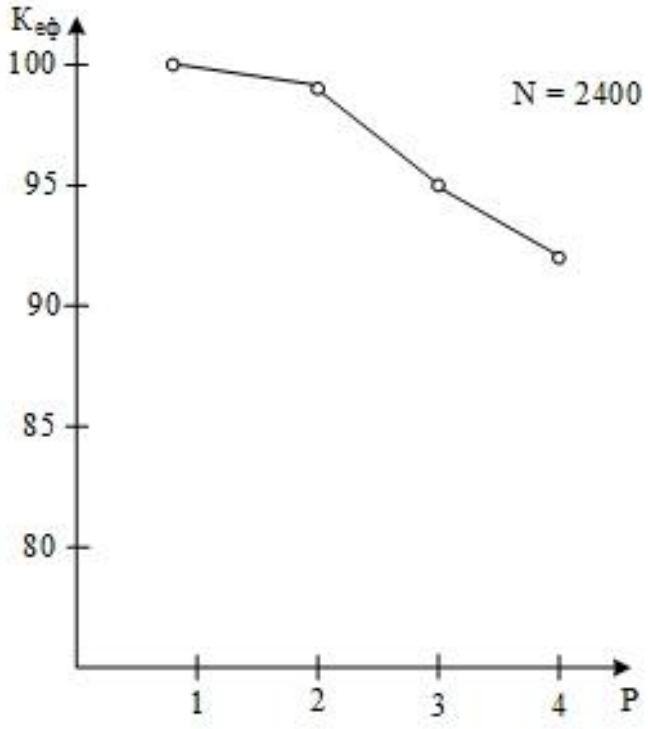

Рис. 2.6. Графік зміни коефіцієнта ефективності $К_{еф}$ залежно від кількості ядер для N = 2400

2.6. Висновки до розділу 2

У даному розділі виконано розробку програми ПРГ1 для ПКС зі СП з використанням мови C#. Тестування програми показало наступне:

- використання багатоядерної ПКС та програми ПРГ1 забезпечує скорочення часу обчислення заданого математичного виразу. Значення $К_{пр}$ знаходиться в межах від 1 до 3. 68;
- максимальне значення $К_{пр}$ забезпечує ПКС з P = 4 та N = 2400;
- мінімальне значення $К_{пр}$ забезпечує ПКС з P = 1 та N = 2400, N = 1600, N = 800;
- зі збільшенням значення N $К_{пр}$ для P = 1 не змінюється, для P = 2, P = 3, P = 4 збільшується;
- значення $К_{еф}$ змінюється в межах від 90% до 100%;
- максимальне значення $К_{еф}$ забезпечує ПКС з P = 1 та N = 800, N = 1600, N = 2400;
- мінімальне значення $К_{еф}$ забезпечує ПКС з P = 4 та N = 800;
- зі збільшенням значення N $К_{еф}$ для P = 1 не змінюється, для P = 2, P = 3, P = 4 зростає.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС З ЛП

У даному розділі розроблено програму ПРГ2 для паралельної комп'ютерної системи з локальною пам'яттю, що відповідає заданому математичному виразу:

A = sort(d*Z) + S*(MO*MK)

Обчислення математичного виразу реалізовано з використанням механізму рандеву мови Ада. Вхідні дані вводяться в ПВВ1, ПВВ2, ПВВ3, а вихідні – в ПВВ1.

Структурна схема паралельної комп'ютерної мережі з локальною пам'яттю зображена в Додатку Г.

3.1. Розробка паралельного математичного алгоритму

Паралельний математичний алгоритм для заданого виразу, розроблений за правилами ярусно-паралельної реалізації задач:

1) Rh = sort(Zh);
2) R2h = mergeSort(Rh, Rh);
3) R = mergeSort(R2h, R2h);
4) $A_H = d*R_H + S*(MO_H*MK)$;

3.2. Розробка алгоритмів процесів

Для даної програми розроблено алгоритм кожного потоку, який показано в таблиці 3.1..

Таблиця 3.1. **Алгоритм роботи потоків**

T1	T2
1) прийняти дані Z_{2H}, MK, d від T2 та S, MO_{3H} від T4; 2) передати дані Z_H, MK, d в T4 та S, MO_{2H} в T2; 3) сортування Rh = sort(Zh); 4) прийняти дані Rh_H від T4; 5) сортування R2h=mergeSort(Rh,Rh);	1) Ввід d; 2) прийняти дані Z_{3H}, MK від T3 та S, MO_{2H} від T1; 3) передати дані Z_{2H}, MK, d в T1 та S, MO_H в T3; 4) сортування Rh = sort(Zh); 5) прийняти дані Rh_H від T3;

Продовження таблиці 3.1.

6) прийняти дані $R2h_{2H}$ від T2; 7) сортування R = mergeSort(R2h, R2h); 8) передати дані R_H в T4 та R_{2H} в T2; 9) Обрахунок $A_H = d*R_H + S*(MO_H *MK)$; 10) прийняти A_H від T4; 11) передати A_{2H} в T2.	6) сортування R2h = mergeSort(Rh, Rh); 7) передати дані $R2h_{2H}$ в T1; 8) прийняти дані R_{2H} від T1; 9) передати R_H в T3; 10) Обрахунок $A_H = d*R_H + S*(MO_H *MK)$; 11) прийняти A_H від T3 та A_{2H} від T1; 12) вивід A;

T3	T4
1) Ввід Z, MK; 2) передати дані Z_{3H}, MK в T2; 3) прийняти дані d, S, MO_H від T2; 4) сортування Rh = sort(Zh); 5) передати дані Rh_H в T2; 6) прийняти дані R_H від T2; 7) Обрахунок $A_H = d*R_H + S*(MO_H *MK)$; 8) передати дані A_H в T2.	1) Ввід S, MO; 2) передати дані S, MO_{3H} в T1; 3) прийняти дані d, Z_H, MK від T1; 4) сортування Rh = sort(Zh); 5) передати дані Rh_H в T1; 6) прийняти дані R_H від T1; 7) Обрахунок $A_H = d*R_H + S*(MO_H *MK)$; 8) передати дані A_H в T1.

3.3. Розробка схеми взаємодії процесів

Графічне зображення взаємодії задач у даній програмі зображено в Додатку Д. На структурній схемі визначаються входи задач, задачі, їх взаємодія. Задано напрям і обсяг даних, що передаються між задачами.

3.4. Розробка програми ПРГ 2

Паралельна програма з локальною пам’яттю реалізована з використанням механізму рандеву мови Ада.

Розроблена програма виконує заданий математичний алгоритм для чисел, векторів, матриць розмірності N та змінною кількістю потоків P.

Усі змінні в програмі– локальні. Входи задач – SetZMKd I SetSMO використовуються для розсилання вхідних даних із потоків. Входи задач – SetR – використовуються для збирання результату іншими потоками відсортованого вектора Z. Входи GetA застосовуються для передачі результату обчислень іншим потокам та виводу даних потоком T(2).

Усі матриці та вектори, що виконуються паралельно, розподілені на однакові частинки H.

Лістинг коду прикріплений у додатку E.

3.5. Результати тестування програми ПРГ 2

Тестування виконано на паралельній обчислювальній системі із характеристиками системи:

- процесор Intel Core i5-7100 ядра по 3.9 GHz, об'єм кеш пам'яті 3 рівня 6MB;
- операційна система Windows 10 Home;
- середовище розробки ObjectAda;

Для тестування обрано такі розмірності векторів і матриць: N = 800, N = 1600, N = 2400. Програму виконано в ПКС із різним значенням кількості ядер P – від 1 до 4. Результати виміру часу відображаються в таблиці 3.2.

Таблиця 3.2. **Час виконання програми для ПРГ2**

N	T1	T2	T3	T4
900	17.73	9.34	6.75	5.07
1800	192.58	98.72	71.59	57.32
2400	489.62	255.01	189.78	148.82

На основі даних таблиці 3.2. обчислено значення коефіцієнтів прискорення, де $К_{пр} = T_1/T_P$. Значення наведені в таблиці 3.3.

Таблиця 3.3. **Значення $К_{пр}$ для ПРГ2**

N	P1	P2	P3	P4
900	1	1.86	2.63	3.49
1800	1	1.95	2.69	3.36
2400	1	1.92	2.58	3.29

На основі даних таблиці 3.3. обчислено значення коефіцієнтів ефективності, де $К_{еф} = К_{пр} / P * 100\%$. Значення наведені в таблиці 3.4.

Таблиця 2.4. **Значення $К_{еф}$ для ПРГ2**

N	T1	T2	T3	T4
900	100 %	93 %	88 %	87 %
1800	100 %	98 %	89 %	84 %
2400	100 %	96 %	86 %	82 %

На підставі таблиць 3.3. та 3.4. побудовано графіки поведінки $К_{пр}$ та $К_{еф}$ залежно від N для ПРГ2.

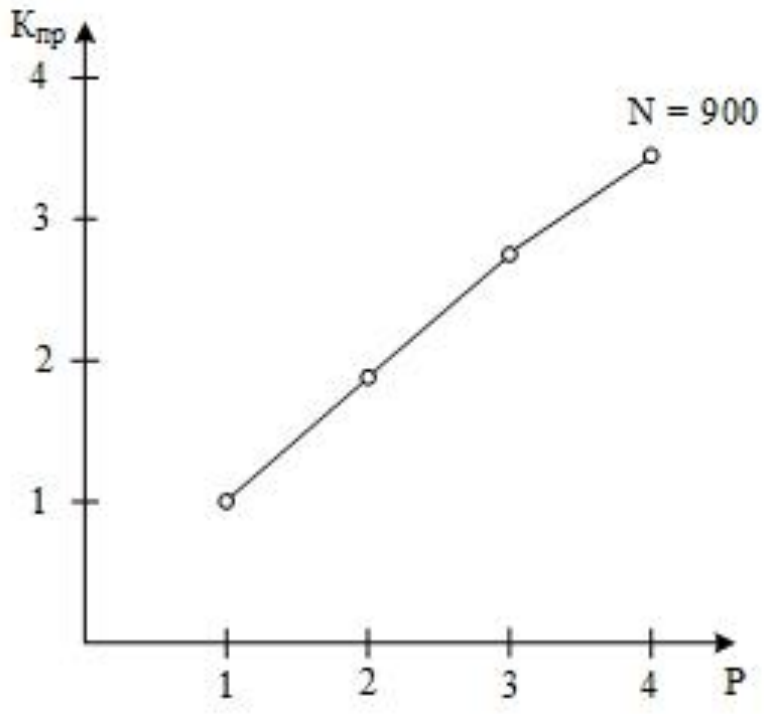

Рис. 3.1. Графік зміни коефіцієнта прискорення $К_{пр}$ залежно від кількості ядер для N = 900

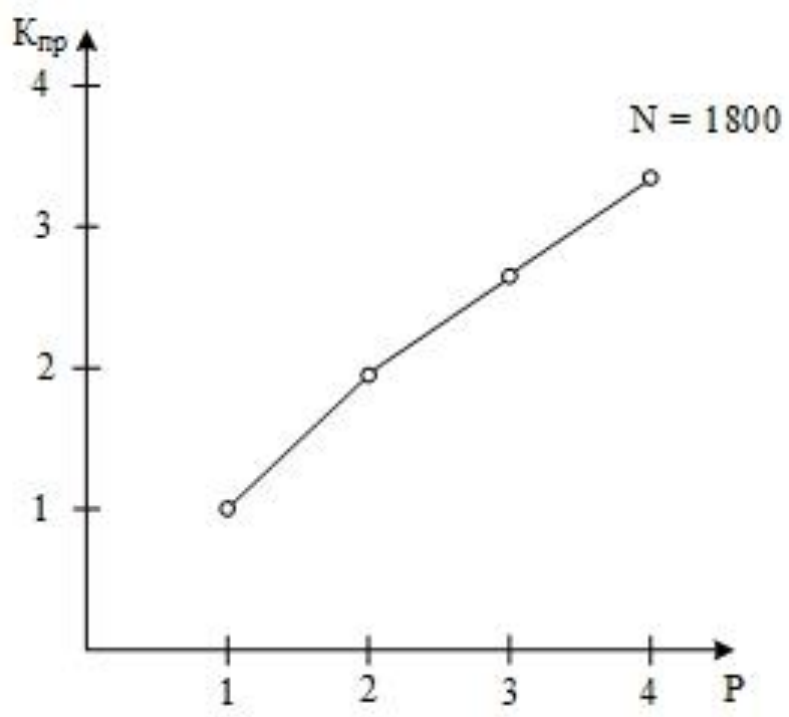

Рис. 3.2. Графік зміни коефіцієнта прискорення $К_{пр}$ залежно від кількості ядер для N = 1800

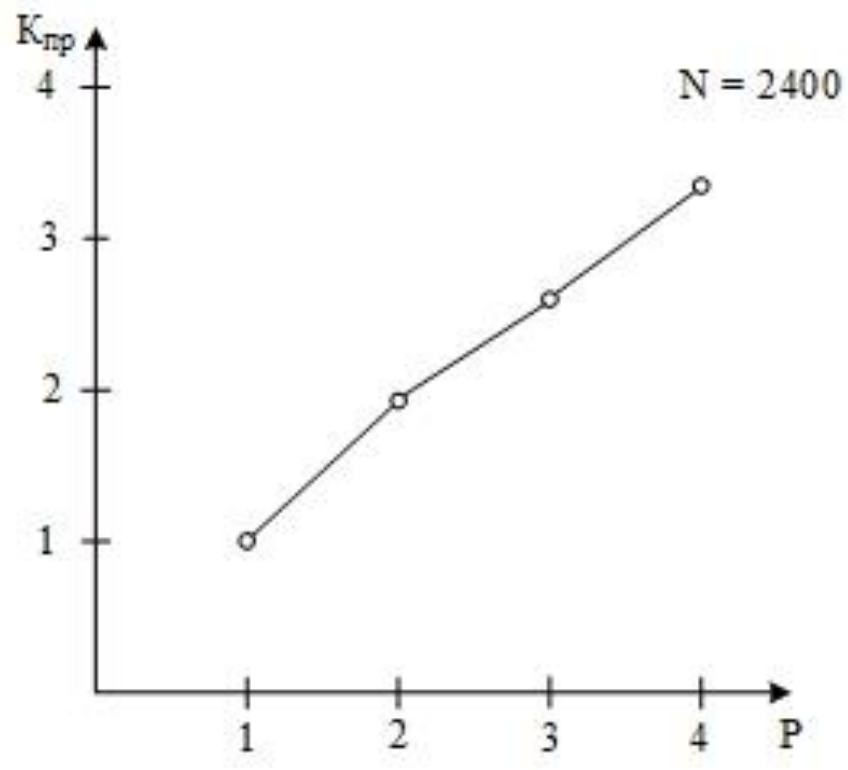

Рис. 3.3. Графік зміни коефіцієнта прискорення $К_{пр}$ залежно від кількості ядер для N = 2400

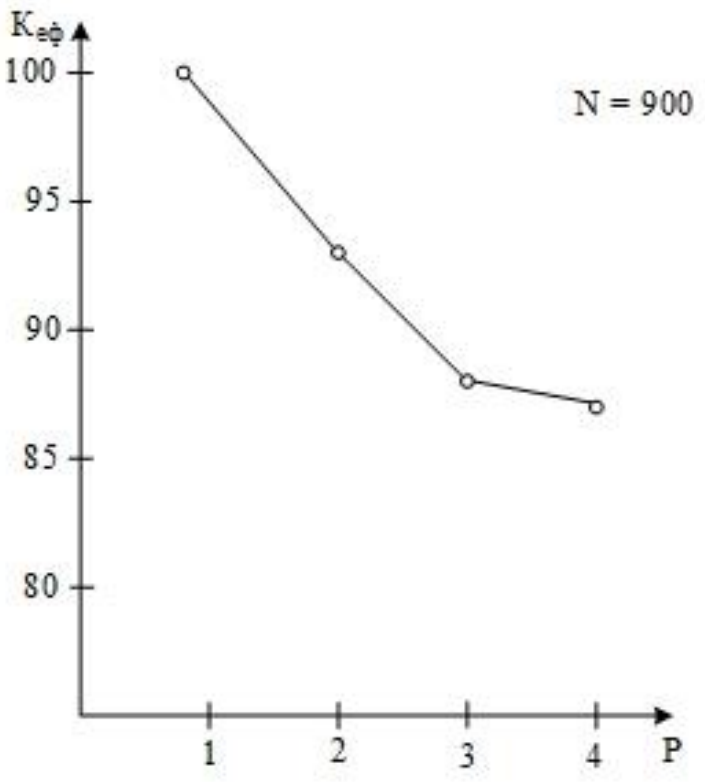

Рис. 3.4. Графік зміни коефіцієнта ефективності $К_{еф}$ залежно від кількості ядер для N = 900

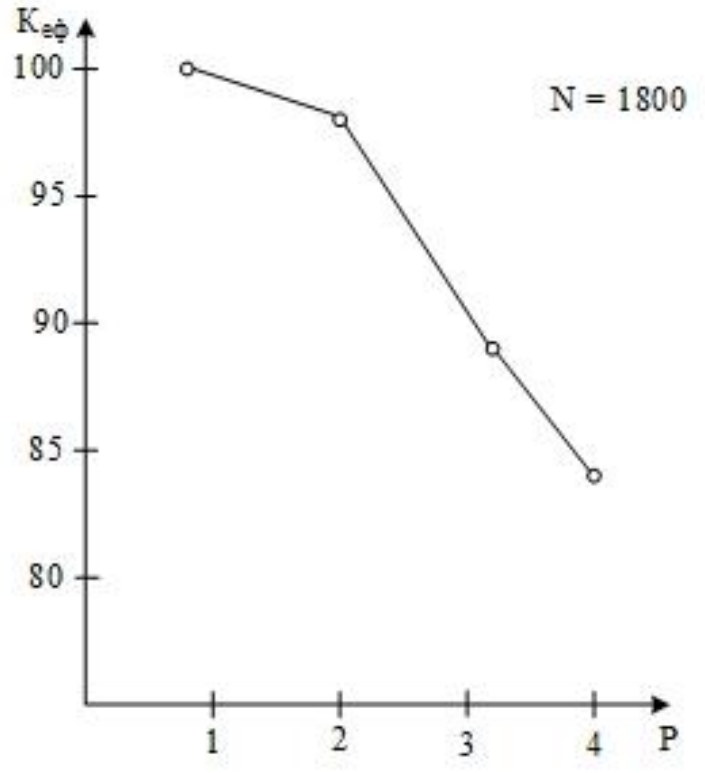

Рис. 3.5. Графік зміни коефіцієнта ефективності $К_{еф}$ залежно від кількості ядер для N = 1800

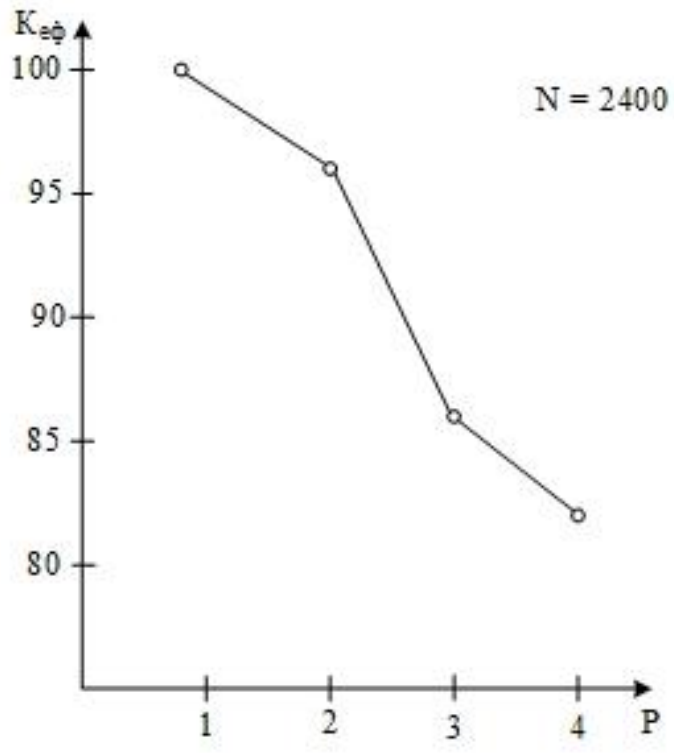

Рис. 3.6. Графік зміни коефіцієнта ефективності $К_{еф}$ залежно від кількості ядер для N = 2400

3.6. Висновки до розділу 3

У даному розділі виконано розробку програми ПРГ2 для ПКС зі ЛП з використанням мови Ада та засобом синхронізації – механізм Рандеву. Тестування програми показало наступне:

- використання багатоядерної ПКС та програми ПРГ2 забезпечує скорочення часу обчислення заданого математичного виразу. Значення $К_{пр}$ знаходиться в межах від 1 до 3.49;
- максимальне значення $К_{пр}$ забезпечує ПКС з P = 4 та N = 900;
- мінімальне значення $К_{пр}$ забезпечує ПКС з P = 1 та N = 900, 1800 чи 2400;
- зі збільшенням значення N $К_{пр}$ для P = 1 не змінюється, для P = 4 зменшується;
- значення $К_{еф}$ змінюється в межах від 82% до 100%;
- максимальне значення $К_{еф}$ забезпечує ПКС з P = 1 та N = 900, 1800 чи 2400;
- мінімальне значення $К_{еф}$ забезпечує ПКС з P = 4 та N = 2400;

- зі збільшенням значення N $К_{еф}$ для P = 1 не змінюється, для P = 4 зменшується.

ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ДО РОБОТИ

1. Проаналізовано, що у будь-яких мовах семафори мають операції створення або конструктори, реалізацію операції P(S) та реалізацію операції V(S), що є основними операціями для використання семафорів.
2. Розроблено програмне забезпечення для вирішення заданої математичної задачі двома різними способами: ПРГ1 зі спільною пам’яттю та ПРГ2 з локальною.
3. Проведено тести результативності ПРГ1 та ПРГ2 на розмірностях N = 900, N = 1800, N = 2400 які показали, що ПРГ1 виконується приблизно в 1.3 рази скоріше, ніж ПРГ2.
4. Показано, що коефіцієнт прискорення та коефіцієнт ефективності приблизно в 1.1 рази більший у ПРГ1, ніж у ПРГ2.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. І. А. Жуков, О. В. Корочкін Паралельні та розподілені обчислення – Київ: «Корнійчук» 2005. – 224 с.
2. Semaphore (Java 2 Platform SE 5.0) [Електронний ресурс] – режим доступу:http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/Semaphore.html
3. Semaphore and SemaphoreSlim [Електронний ресурс] – режим доступу: http://msdn.microsoft.com/en-us/library/z6zx288a(v=VS.110).aspx
4. SemaphoreSlim Class (System.Threading) [Електронний ресурс] – режим доступу: http://msdn.microsoft.com/en-us/library/system.threading.semaphoreslim(v=vs.110).aspx
5. Semaphore Class (System.Threading) [Електронний ресурс] – режим доступу: http://msdn.microsoft.com/en-us/library/system.threading.semaphore(v=vs.110).aspx
6. Корочкин А.В. Ада 95: Введение в программирование. - Киев; Свит, 1998. - 260 с.
7. Операционные системы I [Електронний ресурс] – режим доступу: http://khpi-iip.mipk.kharkiv.edu/library/spo/book/i_g08.html
8. Семафор (информатика) [Електронний ресурс] – режим доступу: http://dic.academic.ru/dic.nsf/enc_tech/1085/%D1%81%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80
9. Synchronization Functions [Електронний ресурс] – ржим доступу: http://msdn.microsoft.com/en-us/library/aa450749.aspx
10. ada.synchronous_task_control [Електронний ресурс] – режим доступу: http://www.infeig.unige.ch/support/ada/gnatlb/a-sytaco.html
11. Реализация семафоров - Архитектура операционной системы UNIX [Електронний ресурс] – режим доступу: http://www.e-reading.org.ua/chapter.php/92786/200/BahArhitektura_operacionnoii_sistemy_UNIX.html
12. Адское программирование. [Електронний ресурс] – режим доступу: http://www.ada-ru.org/V-0.4w/toc_ru.html

13. SemaphoreFullException Class (System.Threading) [Електронний ресурс] – режим доступу: http://msdn.microsoft.com/ru-ru/library/system.threading.semaphorefullexception(v=vs.110).aspx

Додаток А

Структурна схема ПКС зі СП

Додаток В

Лістинг ПРГ1 для ПКС зі СП

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace CSharpThreads
{
    class Program
    {
        //System parameters
        private const int N = 1000;
        private const int P = 4;
        private const int H = N / P;

        //Instruments of interaction
        private static Barrier inputBarrier = new Barrier(P);
        private static Barrier firstSortBarrier = new Barrier(P);
        private static Barrier twoSortBarrier = new Barrier(P/2);
        private static Barrier computeBarrier = new Barrier(P);
        private static EventWaitHandle eventSort = new ManualResetEvent(false);
        private static object objLock = new object();
        private static object objMonitor = new object();

        //Input and output values
        private static Matrix MO;
        private static Matrix MK;
        private static Vector A = new Vector(N, 0);
        private static Vector Z;
        private static Vector R;
        private static Vector[] firsSort = new Vector[4];
        private static Vector[] twoSort = new Vector[2];
        private static Vector S;
        private static volatile int d;

        private static void ThreadFunction(int threadID)
        {
            Console.WriteLine("Thread {0} started!", threadID+1);

            switch(threadID)
            {
                case 0:
                    Z = new Vector(N, 1);
                    break;
                case 1:
                    d = 1;
                    break;
                case 2:
                    MO = new Matrix(N, 1);
                    S = new Vector(N, 1);
                    break;
                case 3:
                    MK = new Matrix(N, 1);
                    break;
            }
            //input barrier
            inputBarrier.SignalAndWait();

            //sort first
            firsSort[threadID] = new Vector(Z, threadID * H, (threadID + 1) * H).Sort();
```

```
//first sort barrier
firstSortBarrier.SignalAndWait();

//two sort
switch (threadID)
{
    case 0:
        twoSort[0] = new Vector().MergeSort(firsSort[0], firsSort[1]);
        break;
    case 2:
        twoSort[1] = new Vector().MergeSort(firsSort[2], firsSort[3]);
        break;
}

//two sort barrier
twoSortBarrier.SignalAndWait();

//final sort
if (threadID == 0)
{
    R = new Vector().MergeSort(twoSort[0], twoSort[1]);
    eventSort.Set();
}
else
{
    eventSort.WaitOne();
}

//shared resources
int di = d;

Vector Si = new Vector(N);
lock (objLock)
{
    Si = new Vector(S);
}

Matrix MKi = new Matrix(N);
Monitor.Enter(objMonitor);
try
{
    MKi = new Matrix(MK);
}
finally
{
    Monitor.Exit(objMonitor);
}

//calculation
Matrix MA = new Matrix(N, 0);
Parallel.For(threadID * H, (threadID + 1) * H, i =>
{
    int buf;
    for (int j = 0; j < N; j++)
    {
        buf = 0;
        for (int k = 0; k < N; k++)
        {
            buf += MO.Data[i, k] * MKi.Data[k, j];
        }
        MA.Data[i, j] = buf;
    }
    buf = 0;
    for (int j = 0; j < N; j++)
    {
```

```
                buf += Si.Data[j] * MA.Data[i, j];
            }
            A.Data[i] = buf + di * R.Data[i];
        });

        //compute barrier
        computeBarrier.SignalAndWait();

        //show result in first thread
        if (threadID == 0) A.Print();

        Console.WriteLine("Thread {0} finished!", threadID+1);
    }

    static void Main(string[] args)
    {
        Console.WriteLine("Main thread started!");

        Parallel.For(0, P, i =>
        {
            Thread thread = new Thread(() => ThreadFunction(i));
            thread.Start();
            thread.Join();
        });

        Console.WriteLine("Main thread finished!");
        Console.ReadKey();
    }
}
   }
```

Додаток Г

Структурна схема ПКС з ЛП

Додаток Д

Структурна схема взаємодії задач ПРГ2

Додаток Е

Лістинг коду ПРГ2 з ЛП

```
with Ada.Text_IO, Ada.Integer_Text_IO, Ada.Calendar;
use Ada.Text_IO, Ada.Integer_Text_IO, Ada.Calendar;
-----------------------Main program-----------------------------
--Programming for parallel computer systems
--Course work part #2. System with local memory.
Ada. Rendezvous
--Krut Vladislav
--NTUU "KPI"
--FICT IO-43
--Task: A = sort(d*Z)+S*(MO*MK)
-------------------------------------------------------------------
procedure Main is
   N: Integer := 4;
   P: Integer := 4;
   H: Integer := N / P;

   StartTime, FinishTime: Time;
   DiffTime: Duration;

   type Vector is array(Integer range<>) of Integer;
   subtype Vector_H is Vector(1..H);
   subtype Vector_2H is Vector(1..2*H);
   subtype Vector_3H is Vector(1..3*H);
   subtype Vector_N is Vector(1..N);

   type Matrix is array(Integer range<>) of Vector_N;
   subtype Matrix_H is Matrix(1..H);
   subtype Matrix_2H is Matrix(1..2*H);
   subtype Matrix_3H is Matrix(1..3*H);
   subtype Matrix_N is Matrix(1..N);
```

```
procedure VectorInput(V: out Vector_N) is

begin
  for i in 1..N loop
    V(i) := 1;
  end loop;
end VectorInput;

procedure VectorOutput(V: in Vector_N) is
begin
  if N<=20 then
    for i in 1..N loop
      Put(V(i));
    end loop;
  end if;
end VectorOutput;

procedure MatrixOutput(M: in Matrix_N) is
begin
  if N<=20 then
    for i in 1..N loop
      New_Line;
      for j in 1..N loop
        Put(M(i)(j));
      end loop;
    end loop;
  else
    Put("Output is too big");
  end if;
end MatrixOutput;

procedure MatrixInput(M: out Matrix_N) is
begin
```

```
  for col in 1..N loop
    for row in 1..N loop
      M(col)(row) := 1;
    end loop;
  end loop;
end MatrixInput;

task Task1 is
  entry SetZMKd(Z :in Vector_2H; MK : in Matrix_N; d: in Integer);
  entry SetSMO(S: in Vector_N; MO :in Matrix_3H);
  entry SetRh(Rh: in Vector_H);
  entry SetR2h(R2h: in Vector_2H);
  entry GetR(R: out Vector_3H);
  entry GetA(A: out Vector_2H);
end Task1;

task Task2 is
  entry SetZMKd(Z: in Vector_3H; MK : in Matrix_N, d: in Integer);
  entry SetSMO(S: in Vector_N; MO: in Matrix_2H);
  entry SetRh(Rh: in Vector_H);
  entry GetR2h(R2h: out Vector_2H);
  entry SetR(R: in Vector_2H);
  entry SetA(A : in Vector_3H);
end Task2;

task Task3 is
  entry SetSMOd(S: in Vector_N; MO: in Matrix_H, d: in Integer);
  entry GetRh(Rh: out Vector_H);
  entry SetR(R: in Vector_H);
  entry GetA(A: out Vector_H);
end Task3;
```

```
task Task4 is
   entry SetdZMK(d: in Integer, Z: in Vector_H, MK: in Matrix_N);
   entry GetRh(Rh: out Vector_H);
   entry SetR(R: in Vector_H);
   entry GetA(A : out Vector_H);
end Task4;

task body Task1 is
   Z_2H: Vector_2H;
   MK1: Matrix_N;
   S1: Vector_N;
   MO_3H:Matrix_3H;
   Rh_H:Vector_H;
   R2h_2H: Vector_2H;
   R:Vector_N;
   A_2H:Vector_2H;
   MOMK:Matrix_N;
   d1:Integer;
   buf, buf1, i1, i2, current :Integer;

begin
   Put_Line("Task 1 started");

   accept SetZMKd(Z: in Vector__2H; MK : in Matrix_N; d: in Integer) do
      Z_2H:= Z;
      MK1:=MK;
      d1:=d;
   end SetZMKd;

   accept SetSMO(S: in Vector_N; MO: in Matrix_3H) do
      MO_3H:= MO;
      S1:=S;
   end SetSMO;
```

```
Task2.SetSMO(S1(1..N), MO_3H(2*H+1..N));
Task4.SetZMKd(Z_H(3*H+1..N), MK1(1..N), d);

for i in 1..H loop
  Rh(i):=Z_H(i);
end loop;

--Sort H
for i in reverse 1..H loop
  for j in 1..(i-1) loop
    if Z_H(j) > Z_H(j+1) then
      buf := Z_H(j);
      Z_H(j):=Z_H(j+1);
      Z_H(j+1):=buf;
    end if;
  end loop;
end loop;

Task4.GetRh(Rh(1..H));

--Merge sort
  i1 := 1;
  i2 := H+1;
  current := 1;
  while i1 <= H and i2 <= 2*H loop
    if Rh(i1) > Rh(i2) then
      Qh(current) := Rh(i2);
      i2 := i2+1;
      current := current+1;
    else
      Qh(current) := Rh(i1);
      i1 := i1+1;
      current := current+1;
```

```
    end if;
  end loop;

  if i1 = H+1 then
    while i2 <= 2*H loop
      Qh(current) := Rh(i2);
      i2 := i2+1;
      current := current+1;
    end loop;
  else
    while i1 <= H loop
      Qh(current) := Rh(i1);
      i1 := i1+1;
      current := current+1;
    end loop;
end if;

for i in 1..H loop
  Rh(i):=Qh(i-H);
end loop;

Task4.SetRh(Rh(1..H));

--Merge sort
  i1 := 1;
  i2 := 2*H+1;
  current := 1;
  while i1 <= H and i2 <= 2*H loop
    if R2h(i1) > R2h(i2) then
      Q2h(current) := R2h(i2);
      i2 := i2+1;
      current := current+1;
    else
```

```
      Q2h(current) := R2h(i1);
      i1 := i1+1;
      current := current+1;
    end if;
  end loop;

  if i1 = 2*H+1 then
    while i2 <= 4*H loop
      Q2h(current) := R2h(i2);
      i2 := i2+1;
      current := current+1;
    end loop;
  else
    while i1 <= 2*H loop
      Q2h(current) := R2h(i1);
      i1 := i1+1;
      current := current+1;
    end loop;
end if;

for i in 2*H+1..4*H loop
  R2h(i):=Q2h(i-2*H);
end loop;

Task2.SetR2h(R2h(1..2*H));

--Merge sort
  i1 := 1;
  i2 := 4*H;
  current := 1;
  while i1 <= 2*H and i2 <= 4*H loop
    if R(i1) > R(i2) then
      R(current) := R(i2);
```

```
      i2 := i2+1;
      current := current+1;
    else
      R(current) := R(i1);
      i1 := i1+1;
      current := current+1;
    end if;
  end loop;

  if i1 = 2*H+1 then
    while i2 <= 4*H loop
      R(current) := R(i2);
      i2 := i2+1;
      current := current+1;
    end loop;
  else
    while i1 <= 4*H loop
      R(current) := R(i1);
      i1 := i1+1;
      current := current+1;
    end loop;
end if;

Task2.GetR(R(H+1..2*H));
Task4.GetR(R(2*H+1..N));

--Computing
for i in 1..H loop
  for j in 1..N loop
    buf1 := 0;
    for k in 1..N loop
      buf1 := buf1 + MO_4H(i)(k)*MK1(k)(j);
    end loop;
```

```
          MOMK(i)(j) := buf1;
      end loop;

      buf1 := 0;
      for l in 1..N loop
        buf1 := buf1 + S1(l)*MOMK(i)(l);
        A_4H(i) := d*R(l) + buf1;
        end loop;
    end loop;

    Task4.SetA(A_H(1..H));
    Task2.GetA(A_2H(1..2*H));

    accept GetA(A : out Vector_N) do
      for i in 1..2*H loop
        A(i+H) := A_H(i);
        A(i+2*H) := A_2H(i+2*H);
      end loop;
    end GetA;

    Put_Line("Task 1 finished");
  end Task1;

  task body Task2 is
    A:Vector_N;
    MK2 : Matrix_N;
    Z_3H: Vector_3H;
    S2: Vector_N;
    MO_2H:Matrix_2H;
    Z_2H: Vector_2H;
    Rh:Vector_H;
    R2h:Vector_2H;
    MOMK:Matrix_N;
```

```
  d:Integer;
  buf, buf1, i1, i2, current :Integer;

begin
  Put_Line("Task 2 started");

  --input data
  IntegerInput(d);

  accept SetZMK(Z :in Vector_3H; MK : in Matrix_N) do
    Z_3H:=Z;
    MK2:=MK;
    d2:=d;
  end SetZMK;

  Task1.SetZMKd(Z_2H(2*H..N), MK2(1..N), d2);

  accept SetSMO(S: in Vector_N; MO :in Matrix_2H) do
        S2:=S;
    MO_2H:=MO;
  end SetSMO;

  Task3.SetSMO(S2(1..N), MO_H(3*H+1..N));

  for i in 1..H loop
    Z_H(i):=Z_H(i);
  end loop;

  --Sort H
  for i in reverse H+1..2*H loop
    for j in 1..(i-1) loop
      if Z_H(j) > Z_H(j+1) then
        buf := Z_H(j);
```

```
        Z_H(j):=Z_H(j+1);
        Z_H(j+1):=buf;
      end if;
    end loop;
  end loop;

  Task3.GetRh(Rh(1..H));

  --Merge sort
    i1 := 1;
    i2 := H+1;
    current := 1;
    while i1 <= H and i2 <= 2*H loop
      if Rh(i1) > Rh(i2) then
        Qh(current) := Rh(i2);
        i2 := i2+1;
        current := current+1;
      else
        Qh(current) := Rh(i1);
        i1 := i1+1;
        current := current+1;
      end if;
    end loop;

    if i1 = H+1 then
      while i2 <= 2*H loop
        Qh(current) := Rh(i2);
        i2 := i2+1;
        current := current+1;
      end loop;
    else
      while i1 <= H loop
        Qh(current) := Rh(i1);
```

```
        i1 := i1+1;
        current := current+1;
      end loop;
end if;

accept SetR2h(R2h: out Vector_2H) do
  R2h := Qh;
end SetR2h;

accept SetR(R: in Vector_H) do
  R_4H := R;
end SetR;

Task3.SetR(R_H(3*H+1..N));
Task1.GetR(R_2H(2*H+1..N));

--Computing
for i in 1..H loop
  for j in 1..N loop
    buf1 := 0;
    for k in 1..N loop
      buf1 := buf1 + MO(i)(k)*MK2(k)(j);
    end loop;
     MOMK(i)(j) := buf1;
  end loop;

  buf1 := 0;
  for l in 1..N loop
    buf1 := buf1 + S2(l)*MOMK(i)(l);
    A(i) := d2*R(l) + buf1;
  end loop;
end loop;
```

```
   Task3.GetA(A(3*H+1..N));
   Task1.GetA(A(2*H+1..3*H));

   --show results
   New_Line;
   Put("A = ");
   New_Line;
   VectorOutput(A);
   New_Line;
   Put_Line("Task 2 finished");

   FinishTime := Clock;
   DiffTime := FinishTime - StartTime;

   Put("Time : ");
   Put(Integer(DiffTime), 1);
   Put_Line("");
end Task2;

task body Task3 is
   MK:Matrix_N;
   MO_H:Matrix_H;
   Z:Vector_3H;
   S:Vector_N;
   A_H:Vector_H;
   Rh_H:Vector_H;
   R_H:Vector_H;
   MOMK:Matrix_N;
   d3:Integer;
   buf, buf1 :Integer;

begin
   Put_Line("Task 3 started");
```

```
VectorInput(Z);
MatrixInput(MK);

Task2.SetZMK(Z(H+1..N), MK(1..N));

accept SetSMOd(MO: in Matrix_H; S: in Vector_N; d: in Integer) do
  MO_H:=MO;
      S3:=S;
     d3:=e;
end SetSMOd;

for i in 1..H loop
  Rh(i):=Z(i+3*H);
end loop;

--Sort H
for i in reverse 2*H+1..3*H loop
  for j in 1..(i-1) loop
    if Z_H(j) > Z_H(j+1) then
      buf := Z_H(j);
      Z_H(j):=Z_H(j+1);
      Z_H(j+1):=buf;
    end if;
  end loop;
end loop;

accept GetR(R: out Vector_H) do
  R := Z_H;
end GetR;

accept SetR(R: in Vector_H) do
  R_H := R;
end SetR;
```

```
  --Computing
  for i in 1..H loop
    for j in 1..N loop
      buf1 := 0;
      for k in 1..N loop
        buf1 := buf1 + MO_H(i)(k)*MK(k)(j);
      end loop;
       MOMK(i)(j) := buf1;
    end loop;

    buf1 := 0;
    for l in 1..N loop
      buf1 := buf1 + S(l)*MOMK(i)(l);
      A_H(i) := d*R_H(l) + buf1;
    end loop;
  end loop;

  accept GetA(A : out Vector_H) do
    A:=A_H(1..H);
  end GetA;

  Put_Line("Task 3 finished");
end Task3;

task body Task4 is
  MK4:Matrix_N;
  MO_H:Matrix_H;
  S4:Vector_N;
  A_H:Vector_H;
  Z_H:Vector_H;
  R_H:Vector_H;
  MOMK:Matrix_N;
  d:Integer;
```

```
  buf, buf1, :Integer;

begin
  Put_Line("Task 4 started");
  VectorInput(Z);
  MatrixInput(MO);

  Task1.SetSMO(MO(H+1..N), S(1..N));

  accept SetdZMK(Z: in Vector_H; MK: in Matrix_N; d: in Integer) do
    Z_H:= Z;
    MK4:=MK;
    d4:=d;
  end SetdZMK;

  --Sort H
  for i in reverse 3*H+1..N loop
    for j in 1..(i-1) loop
      if Z_H(j) > Z_H(j+1) then
        buf := Z_H(j);
        Z_H(j):=Z_H(j+1);
        Z_H(j+1):=buf;
      end if;
    end loop;
  end loop;

  accept GetR(R: out Vector_H) do
    R := Z_H;
  end GetR;

  accept SetR(R: in Vector_H) do
    R_H:= R;
  end SetR;
```

```
    --Computing
    for i in 1..H loop
      for j in 1..N loop
        buf1 := 0;
        for k in 1..N loop
          buf1 := buf1 + MO_H(i)(k)*MK(k)(j);
        end loop;
         MOMK(i)(j) := buf1;
      end loop;

      buf1 := 0;
      for l in 1..N loop
        buf1 := buf1 + S4(l)*MOMK(i)(l);
        A_H(i) := d*R_H(l) + buf1;
      end loop;
    end loop;

    accept GetA(A : out Vector_H) do
      A:=A_H(1..H);
    end GetA;

    Put_Line("Task 4 finished");
  end Task4;

begin
  StartTime := Clock;
end Main;
```