фНаціональний технічний університет України  
«Київський політехнічний інститут»

Кафедра обчислювальної техніки

(повна назва кафедри, циклової комісії)

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)

Студента 3 курсу групи ІП-31

напряму підготовки 6.050103 «Програмна інженерія»

Кобилинський Д.А.

(прізвище та ініціали)

Керівник доцент Корочкін О.В.

Національна оцінка

Кількість балів:

Оцінка: ECTS

**Курсова робота**

з дисципліни «Паралельні та розподілені обчислення»

(назва дисципліни)

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

Київ – 2016 рік

Національний технічний університет України  
«Київський політехнічний інститут»

Факультет (інститут) інформатики та обчислювальної техніки

(повна назва)

Кафедра обчислювальної техніки

(повна назва)

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.050103 «Програмна інженерія»

(шифр і назва)

**Завдання**

на курсову роботу студенту

Кобилинський Дмитро Анатолійович

(прізвище, ім’я, по батькові)

1. Тема роботи «Розробка програмного забезпечення для паралельних комп’ютерних систем».
2. Керівник роботи к. т. н., доцент Корочкін Олександр Володимирович.

(вчене звання, науковий ступінь, прізвище та ініціали)

1. Строк подання студентом роботи 18 травня 2016 р.
2. Вихідні дані для роботи:

* огляд засобів роботи з процесами в бібліотеці PVM;
* математична задача A = B∙max(Z)+ min(Z)∙B∙α(MO∙MK)
* структури ПКС СП та ПКС ЛП;
* бібліотеки програмування: OpenMP, MPI.

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

* огляд засобів роботи з процесами в бібліотеці PVM;
* розробка і тестування програми ПРГ1 для ПКС СП;
* розробка і тестування програми ПРГ2 для ПКС ЛП.

1. Перелік графічного матеріалу:

* структурна схема ПКС СП;
* структурна схема ПКС ЛП;
* схеми алгоритмів процесів і головної програми для ПРГ1;
* схеми алгоритмів процесів і головної програми для ПРГ2.

1. Дата видачі завдання

Календарний план

|  |  |  |
| --- | --- | --- |
| №  з/п | Назва етапів виконання КР | Строк виконання етапів КР |
| 1 | Виконання Розділу 1 | 01.03.2016 |
| 2 | Виконання Розділу 2 | 23.03.2016 |
| 3 | Виконання Розділу 3 | 23.04.2016 |
| 4 | Тестування програм ПРГ1 та ПРГ2 | 10.05.2016 |
| 5 | Оформлення КР | 17.05.2016 |
| 6 | Захист КР | 18.05.2016 |

Студент

(підпис) (прізвище та ініціали)

Керівник роботи

(підпис) (прізвище та ініціали)

ЗМІСТ

[ВСТУП 6](#_Toc451245496)

[РОЗДІЛ 1. ОГЛЯД ЗАСОБІВ РОБОТИ З ПРОЦЕСАМИ В БІБЛІОТЕЦІ PVM 7](#_Toc451245497)

[1.1 Загальна інформація 7](#_Toc451245498)

[1.2 Загальна структура простої програми 9](#_Toc451245499)

[1.3 Створення нової задачі 10](#_Toc451245500)

[1.4 Передача повідомлень 11](#_Toc451245501)

[1.5 Робота з буферами 11](#_Toc451245502)

[1.6 Пакування і розпакування повідомлень 13](#_Toc451245503)

[1.7 Передавання і приймання повідомлень 13](#_Toc451245504)

[1.8 Робота з групами 15](#_Toc451245505)

[1.9 Приклад використання 16](#_Toc451245506)

[1.10 Висновки по розділу 1 16](#_Toc451245509)

[РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ 1 ДЛЯ ПКС СП 17](#_Toc451245510)

[2.1 Розробка паралельного математичного алгоритму. 17](#_Toc451245511)

[2.2 Розробка алгоритмів процесів 18](#_Toc451245512)

[2.3 Розробка схеми взаємодії процесів 18](#_Toc451245513)

[2.4 Розробка програми ПРГ1 20](#_Toc451245514)

[2.5 Тестування ПРГ1 20](#_Toc451245515)

[2.6 Висновки до розділу 2 23](#_Toc451245516)

[РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ 2 ДЛЯ ПКС ЛП 24](#_Toc451245517)

[3.1. Розробка паралельного математичного алгоритму. 24](#_Toc451245518)

[3.2. Розробка алгоритмів процесів 25](#_Toc451245519)

[3.3. Розробка схеми взаємодії процесів ПРГ2 26](#_Toc451245520)

[3.4 Розробка програми ПРГ2 28](#_Toc451245521)

[3.5. Тестування програми ПРГ2 28](#_Toc451245522)

[3.6. Висновки до розділу 3 31](#_Toc451245523)

[СПИСОК ЛІТЕРАТУРИ 33](#_Toc451245524)

[Додаток А. Структурна схема ПКС СП 35](#_Toc451245525)

[Додаток Б. Схема алгоритму головної програми зі вказанням паралельних ділянок для ПРГ1 37](#_Toc451245526)

[Додаток В. Схема алгоритму процесів для програми ПРГ1 39](#_Toc451245527)

[Додаток Г. 41](#_Toc451245528)[Лістинг програми ПРГ1 41](#_Toc451245529)

[Додаток Е. Схема алгоритму головної програми зі вказанням паралельних ділянок для ПРГ2 47](#_Toc451245530)

[Додаток Ж. Схема алгоритму процесів для програми ПРГ2 49](#_Toc451245531)

[Додаток И](#_Toc451245532). [Лістинг програми ПРГ2 51](#_Toc451245533)

## ВСТУП

Паралельні обчислювальні системи - це фізичні комп'ютерні, а так само програмні системи, що реалізують тим чи іншим способом паралельну обробку даних на багатьох обчислювальних вузлах.

Ідея розпаралелювання обчислень базується на тому, що більшість завдань може бути розділене на набір менших завдань, які можуть бути вирішені одночасно. Головна мета використання паралельних обчислень - це підвищення швидкості обчислень за рахунок їх паралельного виконання.

В даній курсовій роботі розглядаються засоби роботи з процесами в одній з найпопулярніших систем паралельного програмування: PVM (Parallel Virtual Machine - паралельна віртуальна машина).

Другий та третій розділи присвячені розробці програми для обчислення математичної задачі в паралельній комп’ютерній системі зі спільною та локальною пам’яттю відповідно.

Для розробки програми обчислення математичної задачі в паралельній комп’ютерній системі зі спільною пам’яттю використана бібліотека OpenMP.

Для розробки програми обчислення математичної задачі в паралельній комп’ютерній системі із локальною пам’яттю використовується реалізація бібліотеки MPI для Java – MPJ Express. Топологія системи – маштабована решітка. Для пересилки даних у такій системи створюється буфер, у який можна помістити об’єкти вектора, матриці, числа, та пересилати їх як одне ціле. Для пересилки необхідно щоб JVM серіалізовувала об’єкти вектора і матриці, а також самого буфера.

Лістинги та алгоритми розроблених програм наведено у додатках.

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

## Загальна інформація

PVM[1] (Parallel Virtual Machine) є побічним продуктом науково-дослідницького проекту, пов’язаного з гетерогенними розрахунками. Головними цілями цього проекту є дослідження гетерогенних розрахунків і розробка зручних, універсальних засобів для організацію гетерогенних розрахунків з використанням комп’ютерів різної архітектури. PVM представляє собою інтегрований набір бібліотек і програмних інструментів для організації гетерогенних розрахунків з використанням зв’язаних між собою обчислювальних пристроїв різноманітних архітектур. Загальна мета PVM – зв’язування цих різноманітних систем для одночасних і паралельних розрахунків.

* Основні принципи PVM:
* Користувацький пул хостів – розрахунок задачі виконується на множині комп’ютерів, яка задається користувачем для запуску даної PVM програми. Як і однопроцесорні системи, так і багатопроцесорні системи(з різними типами пам’яті) можуть бути частиною пулу. Пул може бути змінений під час виконання програми, з нього можуть бути видалені, або додані нові хости(це забезпечує надійність розрахунків під час відмови одного з хостів)
* Прозорий доступ до обладнання – прикладні програми можуть розглядати обладнання як колекцію віртуальних процесорних елементів, або можуть обирати хости з конкретними, найбільш підходящими, апаратними засобами.
* Обчислення на основі процесів – одиниця паралелізму в PVM є задача(найчастіше Unix процес), незалежний послідовний потік управління, який працює в режимі розрахунку або комунікації. Ніякої відповідності задача – процесор PVM встановлює(можливі випадки, коли декілька задач виконуються на одному і тому ж процесорі)
* Модель явної передачі повідомлень – набір задач, кожна з яких виконує свою частину обчислень, явно взаємодіють через відсилання і приймання повідомлень один одному. Розмір повідомлень обмежений лише об’ємом доступної пам’яті.
* Підтримка неоднорідності – система PVM підтримує неоднорідність в плані машин, мереж, програм. Що до передачі повідомлень, PVM дозволяє передавати декілька типів даних між машинами, які мають різне представлення даних.
* Підтримка багатопроцесорних систем – система PVM може використовувати засоби передачі хоста, на якому виконується програма. Таким чином можливе використання переваг обладнання і операційної системи. Деякі розробники представляють власні, оптимізовані для їхніх систем, реалізації PVM, які можуть бути зв’язані з публічної версією PVM.

Система PVM складається з двох частин. Перша частина являє собою демон, який носить назву *pvmd3*, інколи скорчено *pvmd* . Демони знаходяться на всіх комп’ютерах, які входять в склад віртуальної машини PVM. Перед запуском PVM програми користувач має запустити демон і створити віртуальну машину PVM. Користувач може створити декілька віртуальних машин і виконувати декілька програм одночасно.

Інша частина являє собою бібліотеку інтерфейсу PVM. Вона включає в себе набір примітивів, які необхідні для комунікації між задачами і програмами. Ця бібліотека містить набір функції, які викликаються користувачем, для передачі повідомлень, створення задач і їх координації, а також функції для зміни структури віртуальної машини PVM.

Модель розрахунків PVM заснована на тому, що програма складається з декількох задач. Кожна задача відповідає за частину розрахункового навантаження програми. Інколи програму розділяють за функціями, тобто кожна задача виконую свою окрему функцію(введення даних, розрахунок, вивід даних…). Цей метод носить назву функціонального паралелізму. Але більш поширеним способом організації обчислень є паралелізм даних. В цьому методі всі задачі дуже подібні, але кожна з них оброблює лише не велику частину даних.

Система PVM підтримує мови програмування С, С++ та Фортран. Вибір мов програмування був зроблений на основі досліджень, які показали, що дані мови найчастіше використовуються для складних розрахунків.

Для С та С++ прив’язки інтерфейсу PVM реалізовані у вигляді функцій у відповідності з прийнятими стандартами. А якщо точніше, аргументи функцій являють собою значення параметрів і вказівники(при необхідності) а результатом функції є статус виконання виклику функції. Також є набір макросів і функції для визначення типу результату. Для використання PVM в мовах С та С++ використовується бібліотека *libpvm3.a*, що є частиною стандартної поставки.

В мові Фортран прив’язка виконана у вигляді підпрограм, а не функцій. Це пов’язано з тим. що деякі компілятори не можуть надійно підтримувати роботу функції. Тому вводиться додатковий аргумент підпрограм, через який в основну програму повертається результат виклику функцій PVM. Для використання в мові Фортран необхідно мати бібліотеку *libfpvm3.a* і бібліотеку для мови С(*libpvm3.a*).

Всі задачі в PVM, незалежно від обраної мови програмування, мають унікальний цілочисельний ідентифікатор задачі(TID). Повідомлення передаються і отримуються за допомогою TID. TID є унікальним у всій віртуальній машині PVM, і задаються *pvmd*, а не користувачем.

Існують програми, де необхідно використати *групу задач*. І є випадки, коли користувач хоче визначати свої задачі по номерам 0-(p-1), де p – кількість задач. PVM включає концепцію груп, іменованих користувачем. Коли задача приєднується до такої групи, їй присвоюється унікальний номер в межах цієї групи. Згідно з філософією PVM, групи є засобом для об’єднання дуже спільних задач. Різні групи можуть перекриватись, і задачі, які входять в декілька груп, можуть використовувати для передачі повідомлень з однієї групи в іншу.

В загальному розробка програми на PVM виглядає так: користувач створює програму, використовуючи будь яку доступну мову, записує об’єктні файли на хости, які входять до пулу. Ці об’єктні файли є унікальними для різних архітектур. Далі, як правило, користувач запускає одну задачу(головну, або початкову), яка в свою чергу запускає наступні необхідні задачі.

## 1.2 Загальна структура простої програми

Перш за все необхідно включити в програму опис функцій PVM. В мові С це робиться за допомогою директиви препроцесора *include*(*#include "pvm3.h"* ). Також необхідно зберігати TID поточної задачі і MSGTAG(аналог TID для повідомлень). Обидва параметри мають тип int. Якщо використовується головна задача, то вона має містити TID всіх задач, які будуть створені нею.

Для обміну текстовими повідомленнями необхідно використовувати буфер, пам'ять під якого має бути виділена перед початком виконання задачі. В мові С зазвичай буфером є масив з елементів типу char. Розмір масиву обирається в залежності від максимального розміру повідомлення. Інколи, для забезпеченні можливості легкого розширення програми, розмір буферу визначають з запасом.

Для керування правильністю виконання програми необхідно використати ще одну цілочисельну змінну, в яку буде записуватись результат виконання функцій. Викликаючи будь яку функцію, ми зберігаємо результат її виклику. Далі в програмі ми перевіряємо цей результат, і згідно з ним корегуємо виконання програми.

При використанні механізму іменованих груп необхідно зберігати ідентифікатори груп, а також, в залежності від програми, відповідність задач до груп.

## 1.3 Створення нової задачі

Функція для створення нової задачі має сигнатуру:

int numt = pvm\_spawn( char \*task, char \*\*argv, int flag, char \*where, int ntask, int \*tids )

Параметри:

* task – шлях до виконуваного файлу програми. Шлях може бути абсолютним, або відносним до шляху пошуку PVM.
* argv – масив аргументів, яку будуть передані до створеної задачі(якщо підтримується хостом). Якщо необхідності в аргументах немає, параметр має бути NULL.
* flag – ціле число, яке вказує на параметри створення, є сумою наступних параметрів
  + PvmTaskDefault – PVM обирає будь який хост для створення задачі
  + PvmTaskHost – вказати певний хост
  + PvmTaskArch – вказати певну архітектуру
  + PvmMppFront – запустити задачу на хості МПП архітектурі
* where – рядок, який вказує, де запустити задачу. В залежності від flag це може бути назва хоста(ibm1.epm.ornl.gov'), назва архітектури(SUN4). Якщо flag рівний 0, то where ігнорується, і задача буде запущена на будь якому вільному хості
* ntask – кількість задач для запуску
* tids – вказівник на масив, в який будуть записані ідентифікатори створених задач.
* numt – статус виконання створення задач. Якщо значення менше 0, то виникла помилка при створенні, якщо більше 0 але менше ntask, то при створенні деяких задач виникла помилка, в такому випадку користувач повинен перевіряти tids на коди помилок.

Коди помилок:

* PvmBadParam – не правильне значення параметру функції
* PvmNoHost – вказаний хост не є віртуальною машиною
* PvmNoFile – не знайдено виконавчий файл програми
* PvmNoMem – недостатньо пам’яті на хості
* PvmSysErr – pvmd не відповідає на запити
* PvmOutOfRes – недостатньо ресурсів

Приклад використання функції:

numt = pvm\_spawn( "host", 0, PvmTaskHost, "sparky", 1, &tid[0] );

Тут створюється задача з файлу host, без використання аргументів, на архітектурі sparky в кількості 1 штука. Ідентифікатор задачі буде збережено в нульовому елементі масиву tid.

## Передача повідомлень

Засобом взаємодії задач в PVM є обмін повідомленнями. Існує ціла низка функцій для роботи з повідомленнями, і буферами, що використовуються для зберігання повідомлень.

## Робота з буферами

Для ініціалізації відправки повідомлень використовується функція pvm\_initsend.

Сигнатура:

int bufid = pvm\_initsend( int encoding )

Параметри:

* encoding – кодування повідомлень
  + PvmDataDefault – автоматичне кодування даних
  + PvmDataRaw – кодування відсутнє
  + PvmDataInPlace – дані будуть упаковані і передані в буфер задачі, що приймає, минаючи буфер задачі, що передає
* bufid – ідентифікатор створеного буфера

Коди помилок:

* PvmBadParam – неправильний параметр кодування
* PvmNoMem – недостатньо пам’яті для буфера

Для створення нового порожнього буфера використовується функція pvm\_mkbuf.

Сигнатура:

int bufid = pvm\_mkbuf( int encoding )

Параметри:

* encoding – кодування повідомлень
  + PvmDataDefault – автоматичне кодування даних
  + PvmDataRaw – кодування відсутнє
  + PvmDataInPlace – дані будуть упаковані і передані в буфер задачі, що приймає, минаючи буфер задачі, що передає
* bufid – ідентифікатор створеного буфера

Коди помилок:

* PvmBadParam – неправильний параметр кодування
* PvmNoMem – недостатньо пам’яті для буфера

Для знищення буферу використовується функція pvm\_freebuf.

Сигнатура:

int info = pvm\_freebuf( int bufid )

Параметри:

* bufid – ідентифікатор буфера для знищення
* info – код виконання функції, якщо менше 0, виникла помилка

Для отримання активного буферу передавання(приймання) використовується функція pvm\_getsbuf (pvm\_getrbuf).

Сигнатура:

int bufid = pvm\_gets(r)buf ()

Параметри:

* bufid – ідентифікатор активного буфера передавання(приймання)

Для переведення активності з одного буферу передавання(приймання) на інший використовується функція pvm\_setsbuf (pvm\_setrbuf).

Сигнатура:

int oldBuf = pvm\_sets(r)buf (int bufid)

Параметри:

* bufid – ідентифікатор нового буфера передавання(приймання)
* oldBuf – ідентифікатор старого буфера передавання(приймання)

## Пакування і розпакування повідомлень

Всі функції пакування і розпакування повідомлень однотипні, і відрізняються лише типом даних, які вони пакують.

Функція для пакування типу float.

Сигнатура:

int info = pvm\_pkcplx( float \*cp, int nitem, int stride )

Параметри:

* cp – вказівник на масив
* nitem – кількість елементів масиву
* stride – кількість елементів для пропуску при пакуванні

## Передавання і приймання повідомлень

Функція передавання повідомлення

Сигнатура:

int info = pvm\_send( int tid, int msgid)

Параметри:

* tid – TID задачі, до якої буде відправлене повідомлення
* msgid – ідентифікатор повідомлення
* info – статус виконання

Функція масового передавання повідомлення

Сигнатура:

int info = pvm\_mcast( int \*tids, int ntask, int msgid)

Параметри:

* tids – масив TID задач, до яких буде відправлене повідомлення
* ntask – кількість задач в масиві
* msgid – ідентифікатор повідомлення
* info – статус виконання

Функція блокованого приймання повідомлення(задача буде заблокована до моменту прийняття повідомлення). Повідомлення записується в активний буфер.

Сигнатура:

int bufid = pvm\_recv( int tid, int msgtag )

Параметри:

* tid –TID задачі, з якої необхідно прийняти повідомлення
* msgtag – ідентифікатор повідомлення
* info – статус виконання

Функція не блокованого приймання повідомлення

Сигнатура:

int bufid = pvm\_nrecv( int tid, int msgtag )

Параметри:

* tid –TID задачі, з якої необхідно прийняти повідомлення
* msgtag – ідентифікатор повідомлення
* bufid – ідентифікатор буфера, в який буде записано прийняте повідомлення

Функція часового приймання повідомлення

Сигнатура:

int bufid = pvm\_trecv( int tid, int msgtag, struct timeval \*tmout )

Параметри:

* tid –TID задачі, з якої необхідно прийняти повідомлення
* msgtag – ідентифікатор повідомлення
* tmout – часовий інтервал, впродовж якого буде очікуватись повідомлення
* bufid – ідентифікатор буфера, в який буде записано прийняте повідомлення

Функція перевірки на наявність повідомлення

Сигнатура:

int bufid = pvm\_probe( int tid, int msgtag )

Параметри:

* tid –TID задачі, з якої необхідно прийняти повідомлення
* msgtag – ідентифікатор повідомлення
* bufid – ідентифікатор буфера, в який буде записано прийняте повідомлення, при умові що воно вже надійшло

## Робота з групами

Функція для додавання задачі до групи

Сигнатура:

int inum = pvm\_joingroup( char \*group )

Параметри:

* group – назва групи
* inum – номер задачі в групі

Функція для покидання групи

Сигнатура:

int info = pvm\_lvgroup( char \*group )

Параметри:

* group – назва групи
* info – статус виконання функції

## Приклад використання

## pvm\_initsend(PvmDataDefault);

## pvm\_pkint(&num\_data, 1, 1);

pvm\_pkint(&a[num\_data\*i], num\_data, 1);

pvm\_send(task\_ids[i], 4);

В даному прикладі ми ініціалізували стандартне кодування повідомлень, запакували одне ціле число, масив цілих чисел, і відправили повідомлення до задачі task\_ids[i] з ідентифікатором повідомлення 4.

## Висновки по розділу 1

Виконано загальний огляд бібліотеки PVM. Визначено, що PVM став результатом дослідницьких робіт з гетерогенних обчислень. Показано, що PVM має засоби для організації обчислень методом відправки повідомлень; має гнучку структуру, що дозволяє будувати неоднорідні системи; має засоби для організації розрахунків з використанням машин різноманітної архітектури.

Виконано огляд моделі задач PVM. Визначено, що задачі є одиницею паралелізму в PVM. Показано, що задачі мають гнучку систему обміну повідомленнями і об’єднання в групи.

На основі огляду системи обміну повідомленнями, можна зробити висновок, що PVM має всі необхідні засоби для створення паралельних програм будь якої складності; PVM має необхідні засоби для вирішення будь яких задач синхронізації.

# РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ 1 ДЛЯ ПКС СП

У даному розділі розроблюється програма ПРГ1 для системи зі спільною пам’яттю, що відповідає технічному завданню, представленому на малюнку нижче.

A = B∙max(Z)+ min(Z)∙B∙α(MO∙MK)

**ОП**

0

P-1

…

## 2.1 Розробка паралельного математичного алгоритму.

Паралельний математичний алгоритм відповідно до рекомендованої методичної літератури [2] можна подати у вигляді наступних двох етапів:

1. *minZi = min(ZH)i = ;*

*minZ = min(minZ, minZi), i = ;*

1. *maxZi=max(ZH),i=  
   maxZ = max(maxZ, maxZi), i =*
2. *MAH = MO∙MKH;*
3. *AH = Bi∙maxZi+ minZi∙B∙αiMAH, i = ,*

де:

* H = N/P;
* AH – H рядків вектора А;
* MKH – H рядків матриці MK;
* ZH – H рядків вектора Z;
* MAH – H рядків матриці MA
* MKH – H рядків матриці MK

Спільні ресурси: α, B, min,max, MO,MA

## 2.2 Розробка алгоритмів процесів

Оскільки розроблюване програмне забезпечення є масштабованим і працює на системі із кількістю процесорів P≥2, то зручним варіантом реалізації є написання єдиного алгоритму для всіх задач.

|  |  |
| --- | --- |
| Задачі Т(0) – Т(P-1) | |
| Крок алгоритму | КД, Бар’єри |
| 1. Якщо tid = 0 Ввести α, B, MK, MO,Z. |  |
| 2. Бар’єр для усіх задач. Синхронізація по вводу. | Бар’єр |
| 3. Обчислення *minZi = min(Z), i =*  4. Обчислення *maxZi = max(Z), i =* |  |
| 5.Обчислення *minZ = min(minZ, minZi),  maxZ = max(maxZ, maxZi), i =* | КД |
| 6. Бар’єр для усіх задач. Синхронізація обчислень minZ,maxZ | Бар’єр |
| 7. Копіювати MOi=MO, *minZi = minZ, maxZi = maxZ , αi = α, Bi = B* | КД |
| 8. Обчислення MAH = MOi∙MKH |  |
| 9. Бар’єр для усіх задач. Синхронізація обчислень MA | Бар’єр |
| 10. Обчислення *AH = Bi∙maxZi+ minZi∙B∙αiMAH, i =* |  |
| 11. Бар’єр для усіх задач. Синхронізація обчислень A | Бар’єр |
| 12. Якщо tid = 0 вивести результат AH. | ­ |

## 2.3 Розробка схеми взаємодії процесів

На основі алгоритму для всіх задач, наведеному в попередньому розділі, було розроблена структурна схема взаємодії задач (рис. 2.1). Вона дозволяє наочно контролювати бар’єри та критичні ділянки. Крім того, на структурній схемі уводяться також замок, та критична секція, що будуть використовуватись в програмі.

Для демонстрації взаємодії між задачами вибрано дві задачі: T(0), T(i) (i = ). Задача T(0) вводить дані, тому з нею взаємодіють всі інші (синхронізація по вводу). Задачі T(i) (i = ) виконують лише обчислення і синхронізуються зі всіма іншими задачами по обчисленню. Задача T(0) виводить результат обчислень.



Рис. 2.1 Структурна схема взаємодії задач

## 2.4 Розробка програми ПРГ1

Програма написана на мові C++ з використанням бібліотеки OpenMP та складається з трьох модулів: основного СourseWork\_PP2.cpp, та допоміжного operations.cpp і файлу заголовків operations.h.

Основний модуль СourseWork\_PP2.cpp містить одну функцію:

* main – точка входу в програму. Формує ідентифікатори tid задач, запускає задачі, а також вимірює час виконання elapsedTime програми ПРГ1.

В цьому модулі об’явлена така змінна: P – налаштування кількості процесорів.

Файл заголовків operations.h. представляє заголовки службових функцій (таких як ввід/вивід, копіювання), та типи вектора і матриці. У ньому також об’явлена змінна N що визначає розмірність матриць і векторів.

Допоміжний модуль operations.cpp реалізовує функції об’явлені у файлі заголовків operations.h.

Лістинг програми ПРГ1 наведено у додатку А.

## 2.5 Тестування ПРГ1

Для тестування використовувалась паралельна обчислювальна система з наступними апаратними характеристиками:

* процесор AMD Phenom II x6 1055T Processor 2.80 GHz
* оперативна пам’ять 3,25 ГБ

В якості програмного забезпечення виступали:

* операційна система: Microsoft Windows 10
* середовище розробки і компіляції C++ програми: Microsoft Visual Studio Premium 2013.

Для вимірювання часу виконання програми використовувався високоточний таймер з бібліотеки OpenMP.

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



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



Результати тестування і проведених досліджень ефективності розробленої програми наведено в таблицях 2.1 – 2.3.

Таблиця 2.1 Час виконання програми ПРГ1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Т1, мс | Т2, мс | Т3, мс | Т4, мс | Т5, мс | Т6, мс |
| 900 | 3606 | 1808 | 1328 | 1045 | 851 | 786 |
| 1800 | 29169 | 14567 | 10407 | 7952 | 6383 | 5530 |
| 2400 | 68163 | 34222 | 24637 | 18724 | 14979 | 12548 |

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

Таблиця 2.2 Коефіцієнти прискорення для програми ПРГ1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Кількість процесорів (P) | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 900 | 1 | 1,99 | 2,72 | 3,45 | 4,24 | 4,59 |
| 1800 | 1 | 2 | 2,8 | 3,67 | 4,57 | 5,27 |
| 2400 | 1 | 1,99 | 2,77 | 3,64 | 4,55 | 5,43 |

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

Таблиця 2.3 Коефіцієнти ефективності для програми ПРГ1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Кількість процесорів (P) | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 900 | 100,00% | 99,74% | 90,52% | 86,31% | 84,75% | 76,46% |
| 1800 | 100,00% | 100,00% | 93,43% | 91,70% | 91,40% | 87,92% |
| 2400 | 100,00% | 99,59% | 92,22% | 91,01% | 91,01% | 90,53% |

Використовуючи таблиці Таблиця 2.2‑Таблиця 2.3 побудовано графік(pис. 2.2 ) зміни коефіцієнтів прискорення і ефективності в залежності від *N* і .

Рис. 2.2 – Графік зміни коефіцієнту прискорення програми ПРГ1 в залежності від кількості ядер

Рис. 2.3 – Графік зміни коефіцієнту ефективності програми ПРГ1 в залежності від кількості ядер

**2.6 Висновки до розділу 2**

Виконано розробку програми ПРГ1 для ПКС СП з використанням мови С++ і засобів синхронізації з бібліотеки OpenMP. Тестування програми ПРГ1 показало наступне:

* Використання багатоядерної ПКС та програми ПРГ1 забезпечує

скорочення часу обчислення заданої математичної задачі при збільшенні P для всіх N, як показано у табл. 2.1. Значення Kn

лежить в межах від 1,99 до 5,43, як показано у табл. 2.2.

* Максимальне значення Kn = 5,43 забезпечує ПКС з P = 6 та N = 2400.
* Мінімальне значення Kn = 1,99 виявлено у ПКС з P = 2 та N = 2400,

N = 900.

* З ростом N для усіх значень P Kn лінійно зростає, як показано на рис.

2.2.

* Значення Ke змінюються від 76,46% до 100%, як показано у

табл.2.3.

* Найефективніше програма ПРГ1 використовує ПКС з P = 2 та N = 1800

при цьому Ke = 100%.

* Найнижча ефективність використання ПКС програмою ПРГ1

виявлена при P = 6, N = 900, як зображено на рис. 2.3.

* Зі зростанням P від 2 до 6 Ke падає від 99,74% до 76,46% при N = 900.
* Зі зростанням P від 2 до 6 Ke залишається практично однаковим,

змінюючись лише від 93,43% до 87,92% для N = 1800 та від 92,22% до

90,53% для N = 2400.

# РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ 2 ДЛЯ ПКС ЛП

У даному розділі розроблюється програма ПРГ2 для системи з локальною пам’яттю, що відповідає технічному завданню, представленому на малюнку нижче.

A = B∙max(Z)+ min(Z)∙B∙α(MO∙MK)



## 3.1. Розробка паралельного математичного алгоритму.

1. *уi=max(ZH),i=  
   у = max(у, уi), i =*
2. *хi = min(ZH)i = ;*

*х = min(х, хi), i = ;*

1. *MAH = MO∙MKH;*
2. *AH = Bi∙уi+ хi∙B∙αiMAH, i =*

H = N/P;

AH – H рядків вектора А;

MKH – H рядків матриці MK;

ZH – H рядків вектора Z;

MAH – H рядків матриці MA

MKH – H рядків матриці MK

## 3.2. Розробка алгоритмів процесів

**Алгоритм процесу Т(0)**

* 1. Введення Z,MO,a;
  2. Упакувати ZH,MO,a;
  3. Надіслати пакети всім процесам (враховуючи топологію);
  4. Отримати пакет з упакованими від Т(p-1);
  5. Розпакувати дані;
  6. Обчислити y0= min();
  7. Надіслати y0 процесу T(p-1);
  8. Отримати y від T(p-1);
  9. Обчислити x0= min();
  10. Надіслати x0 процесу T(p-1);
  11. Отримати x від T(p-1);
  12. Обчислити *AH = Bi∙у+ х∙B∙αiMAH*;
  13. Надіслати процесу Т(p-1) (враховуючи топологію).

**Алгоритм процесу Т(j)**

1. Отримати пакети з упакованими від Т(p-1) та ZH,MO,a від Т0;
2. Надіслати отримані пакети процесам (враховуючи топологію);
3. Розпакувати дані;
4. Обчислити y0 = min(ZH);
5. Надіслати y0 процесу T(p-1);
6. Отримати y від T(p-1);
7. Обчислити x0 = max(ZH);
8. Надіслати x0 процесу T(p-1);
9. Отримати x від T(p-1);
10. Обчислити *AH = Bi∙у+ х∙B∙αiMAH*;
11. Надіслати процесу Т(p-1)(враховуючи топологію).

**Алгоритм процесу Т(p)**

1. Введення MK,B;
2. Упакувати MKH,;
3. Надіслати пакети всім процесам (враховуючи топологію);
4. Отримати пакет з упакованими ZH,MO,a з процесу Т0;
5. Розпакувати дані;
6. Обчислити yp-1= min();
7. Обчислити y = min(y, yp-1);
8. Отримати yj від процесів Tj;
9. Обчислити y = min(y,yj);
10. Надіслати y всім процесам (враховуючи топологію);
11. Обчислити xp-1= max();
12. Обчислити x = max(x, xp-1);
13. Отримати xj від процесів Tj;
14. Обчислити x = max(x,xj);
15. Обчислити *AH = Bi∙у+ х∙B∙αiMAH*;
16. Отримати від процесів Tj (враховуючи топологію);
17. Виведення А.

## 3.3. Розробка схеми взаємодії процесів ПРГ2

На основі алгоритму для всіх задач, розглянутого в попередньому розділі, були розроблені структурні схеми взаємодій задач .

На схемах введені такі умовні позначення для комунікації процесів через використання механізму повідомлень:

Т(0..p-1) – ранг процесу;

Alltoallv – метод колективної передачі даних між процесами. Дозволяє одночасно ініціювати передачу даних, що знаходяться в процесах 0, p-1, між усіма процесами;

Reduce - метод колективної передачі даних між процесами. Дозволяє ініціювати передачу мінімальних значень діапазонів вектору з кожного процесу в процес з рангом p-1 та виконати операцію MPI.MIN/MAX над автоматично сформованим масивом цих значень;

Bcast - метод колективної передачі даних між процесами. Дозволяє передати дані від одного процесу всім іншим враховуючи топологію (передається мінімальне значення вектору Z від процесу p-1 іншим процесам);

Gather - метод колективної передачі даних між процесами. Дозволяє отримати дані від всіх процесів та автоматично їх впорядкувати (використовується для отримання часткових результатів A процесом з рангом 0 від інших процесів).



Рис 3.1.Схема взаємодії задач

## 3.4 Розробка програми ПРГ2

Програма написана на мові Java з використанням реалізації бібліотеки MPI для Java [7], та складається з 4 класів.

Розробка проведена згідно алгоритмів задач, які наведені у підрозділі 3.2, та згідно схеми взаємодії задач, яка зображена на рис. 3.1.

MatrixOperations сервісний клас, який містить методи для виконання математичних операцій над матрицями, векторами та числами згідно завдання;

DataPack реалізує методи що дозволяють встановлювати та повертати значення матриць та векторів

DataPackBuilder реалізує методи що дозволяють виконати пакування данних для передачі між процесами

Main - головний клас, в якому відбувається ініціалізація бібліотеки MPI з параметрами, переданими через командну строку, а також виконання алгоритму взаємодії процесів для вирішення математичного виразу згідно завдання. Для взаємодії процесів створюється декартова топологія (відображає зв’язки в решітці) за допомогою засобів MPI, а також колективних операцій обміну даними між процесами в топології.

## 3.5. Тестування програми ПРГ2

Для тестування ПРГ2 використовувалось те ж саме апаратне і програмне забезпечення, що і для програми ПРГ1.

Для визначення коефіцієнтів прискорення та ефективності ПРГ2 проведений ряд експериментів із різними розмінностями операндів (N = 900, 1800, 2400) і різною кількістю працюючих ядер ( P = 1, 2, 4, 6).

Для обчислення часу роботи використовувалась функція класу System з назвою currentTimeMillis(), яка повертає поточний час у мілісекундах. Час роботи системи вимірювався як різниця між поточним часов завершення та початку обчислень.

Результати тестування і проведених досліджень ефективності розробленої програми наведено в таблицях Таблиця 3.1‑

Таблиця 3.3.

Таблиця 3.1 – Час виконання програми ПРГ2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N |  |  |  |  |  |  |
| 800 | 2921 | 1474 | 1030 | 921 | 879 | 633 |
| 1600 | 50421 | 25425 | 17188 | 14777 | 14204 | 10082 |
| 2400 | 177053 | 89314 | 59997 | 51551 | 49375 | 34780 |

На основі даних із таблиці Таблиця 3.1 виконано розрахунок значень коефіцієнтів прискорення, які наведені в таблиці Таблиця3.2.

Таблиця3.2 – Коефіцієнти прискорення для програми ПРГ2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Кількість процесорів (P) | | | |
| N | 1 | 2 | 4 | 6 |
| 800 | 1 | 1,89 | 2,65 | 3,54 |
| 1600 | 1 | 1,99 | 3,24 | 4,79 |
| 2400 | 1 | 1,93 | 3,27 | 4,63 |

Коефіцієнти ефективності (таблиця

Таблиця 3.3) обчислено за даними таблиці Таблиця3.2.

Таблиця 3.3 – Коефіцієнти ефективності для програми ПРГ2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Кількість процесорів (P) | | | |
| N | 1 | 2 | 4 | 6 |
| 800 | 100,00% | 94,52% | 65,85% | 58,58% |
| 1600 | 100,00% | 98,99% | 81,94% | 79,88% |
| 2400 | 100,00% | 98,09% | 82,72% | 75,90% |

Використовуючи таблиці 3.2 – 3.3 побудовано графіки зміни коефіцієнтів прискорення і ефективності, а також зміни часу роботи програми в залежності від *N* і P.

Рис 3.2. Графік зміни коефіцієнту прискорення програми ПРГ2 в залежності від кількості ядер

Рис 3.3 – Графік зміни коефіцієнту ефективності програми ПРГ2 в залежності від кількості ядер

## 3.6. Висновки до розділу 3

Виконано розробку програми ПРГ2 для ПКС ЛП з використанням мови програмування Java і бібліотеки реалізації бібліотеки MPI для цієї мови. Тестування програми показало наступне:

* Використання багатоядерної ПКС та програми ПРГ2 забезпечує скорочення часу обчислення заданої математичної задачі. Із збільшенням кількості процесорів у ПКС ЛП час роботи ПРГ2 зменшується, як показано у табл. 3.1. Значення Kn лежать в межах від 1,89 до 4,79, як показано у табл. 3.2.
* Максимальне значення Kn = 4,79 забезпечує ПКС з P = 6 та N = 1600. Динаміка зміни Kn для ПКС з такими параметрами зображена на рис. 3.4.
* Мінімальне значення Kn = 1,89 виявлено у ПКС з P = 2 та N = 900. Динаміка зміни Kn для ПКС з такими параметрами зображена на рис. 3.2
* З ростом N Kn лінійно зростає для всіх значень, як показано на рис.3.2.
* Значення Ke змінюється від 58,58% до 98,99%, як показано у табл. 3.3.
* Найефективніше програма ПРГ2 використовує ПКС з P = 2 та N = 1800, при цьому Ke = 98,95%.
* Найнижча ефективність використання ПКС програмою ПРГ2 виявлена при P = 6 та N = 900, як показано на рис. 3.5.
* Зі зростанням P від 1 до 6 Ke лінійно спадає, як показано на рис. 3.3.

Враховуючи значення коефіцієнтів прискорення та ефективності, а також часу виконання програми, можна зробити висновок про доцільність використання бібліотеки MPI при розробці програмного застосунку для ПКС.

## ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ПО РОБОТІ

Виконано загальний огляд бібліотеки PVM. Визначено, що PVM став результатом дослідницьких робіт з гетерогенних обчислень. Показано, що PVM має засоби для організації обчислень методом відправки повідомлень; має гнучку структуру, що дозволяє будувати неоднорідні системи; має засоби для організації розрахунків з використанням машин різноманітної архітектури.

Виконано огляд моделі задач PVM. Визначено, що задачі є одиницею паралелізму в PVM. Показано, що задачі мають гнучку систему обміну повідомленнями і об’єднання в групи.

На основі огляду системи обміну повідомленнями, можна зробити висновок, що PVM має всі необхідні засоби для створення паралельних програм будь якої складності; PVM має необхідні засоби для вирішення будь яких задач синхронізації.

Різниця у часі виконання програм ПРГ1 та ПРГ2 пояснюється використанням абсолютно різних технологій. Оскільки ПРГ2 використовує віртуальну JVM, системі необхідний час на трансляцію комманд JVM.

Значний вплив на різницю у часі роботи ПРГ1 і ПРГ2 впливає складна топологія системи, яку використовує ПРГ2. Це пояснюється тим, що збільшується час на передачу даних між процесорами, і як наслідок, це впливає на загальний час виконання ПРГ2.

Використання неблокуючих пересилок у ПРГ2, де це дозволяє алгоритм ПРГ2 дещо зменшив час роботи ПРГ2.

Використання механізму серіалізації об’єктів у Java, що необхідно для пересилки даних у ПРГ2 між процесорами, негативно впливає на час роботи програми.

Алгоритм роботи для ПРГ1 значно легший і при проектуванні і при реалізації, ніж у ПРГ2. У випаду системи з локальною пам’яттю, велика кількість повідомлень різного розміру ускладнюють систему з пересилкою повідомлень.

## СПИСОК ЛІТЕРАТУРИ

1. Geist A. PVM. Parallel Virtual Machine. A Users' Guide and Tutorial for Network Parallel Computing / Geist A., Dongarra J., Beguelin A.,1965 – 299
2. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення: Навч. посібник [Текст]. – К.: Корнійчук, 2005. – 226 с. – ISBN 996-7599-36-1.
3. Современные операционные системы. Планирование заданий, процессов и потоков [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/studies/courses/631/487/lecture/11055?page=4 – дата звернення 29.02.2016. – Назва з екрану
4. The OpenMP® API specification for parallel programming[Електронний ресурс]. - Режим доступу: http://openmp.org/wp/ – дата звернення 22.03.2016. – Назва з екрану
5. Openmp (включение поддержки OpenMP 2.0) [Електронний ресурс]. -  
   <https://msdn.microsoft.com/ru-ru/library/fw509c3b.aspx?f=255&MSPPError=-2147217396> – дата звернення 22.03.2016. – Назва з екрану
6. OpenMP in Visual C++ [Електронний ресурс]. -https://msdn.microsoft.com/ru-ru/library/tt15eb9t.aspx – дата звернення 22.03.2016. – Назва з екрану
7. MPJ Express Java-Docs[Електронний ресурс]. — Режим доступу <http://mpj-express.org/docs/javadocs/index.html>
8. The OpenMP® API specification for parallel programming[Електронний ресурс] .— Режим доступу <http://openmp>.org/wp/openmp-specifications/
9. Open MPI v1.8.4 documentation [Електронний ресурс] .—Режим доступу: <http://www.open-mpi.org/doc/v1.8/>
10. MPJ Express Java-Docs[Електронний ресурс]. — Режим доступу <http://mpj-express.org/docs/javadocs/index.html>

**Додатки**

**Додаток А. Структурна схема ПКС СП**

**Додаток Б. Схема алгоритму головної програми зі вказанням паралельних ділянок для ПРГ1**

**Додаток В. Схема алгоритму процесів для програми ПРГ1**

## Додаток Г.

## Лістинг програми ПРГ1

*CourseWork\_PP2.cpp*  
/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\*

\* Паралельне програмування

\* Курсова робота. ПРГ1. Бібліотека OpenMP \*

\*

\* Завдання: A = max(Z)\*B + min(Z)\*alfa\*B\*(MO\*MK)

\*

\* Файл courseWork\_PP2.cpp

\* Автор Кобилинський Дмитро

\* Група ІП-31

\* Дата 20.03.16

\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\*/

#include "stdafx.h"

#include "omp.h"

#include <windows.h>

#include <iostream>

#include "operations.h"

#include <ctime>

#include <clocale>

using namespace std;

#pragma comment(linker, "/stack:160000000")

const int P = 4;

int main()

{

vector A = new int[N], B, Z;

matrix MO, MK, MA;

int alfa;

int minZ = 9999999999;

int maxZ = 0;

const int H = N / P;

omp\_lock\_t lock\_Copy;

double start\_time = omp\_get\_wtime();

setlocale(LC\_ALL, "Russian");

omp\_init\_lock(&lock\_Copy);

omp\_set\_num\_threads(P);

#pragma omp parallel

{

int tid = omp\_get\_thread\_num();

MA = initMatrix();

cout << "Задача " << tid << " стартувала" << endl;

switch (tid)

{

case 0:

alfa = 1;

B = inputVector(1);

MK = inputMatrix(1);

Z = inputVector(1);

MO = inputMatrix(1);

break;

}

#pragma omp barrier

int minZid = 9999999999;

for (int i = tid\*H; i < (tid + 1)\*H; i++)

{

if (Z[i]< minZid)

{

minZid = Z[i];

}

}

#pragma omp barrier

int maxZid = 0;

for (int i = tid\*H; i < (tid + 1)\*H; i++)

{

if (Z[i]> maxZid)

{

maxZid = Z[i];

}

}

#pragma omp critical

{

minZ = min(minZ, minZid);

maxZ = max(maxZ, maxZid);

}

#pragma omp barrier

matrix MOid;

vector Bid;

vector Eid;

int alfaId;

omp\_set\_lock(&lock\_Copy);

MOid = copyMatrix(MO);

alfaId = alfa;

minZid = minZ;

maxZid = maxZ;

Bid = copyVector(B);

omp\_unset\_lock(&lock\_Copy);

#pragma omp barrier

int sum;

for (int i = tid \* H; i < (tid + 1) \* H; i++) {

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

sum = 0;

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

sum += MOid[i][j] \* MK[j][z];

}

MA[i][j] = sum;

}

}

#pragma omp barrier

int buf;

for (int i = tid\*H; i < (tid + 1)\*H; i++)

{

A[i] = 0;

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

{

A[i] += minZid\*Bid[j] \* MA[j][i];

}

A[i] = A[i] \* alfaId;

}

for (int i = tid\*H; i < (tid + 1)\*H; i++)

{

A[i] += maxZid\*B[i];

}

#pragma omp barrier

if (tid == 0)

{

output(A);

}

cout << "Задача " << tid << " завершилась" << endl;

}

double end\_time = omp\_get\_wtime();

double elapsedTime = end\_time - start\_time;

cout << "Час роботи: " << elapsedTime << endl;

cin.get();

return 0;

}

*Operations.cpp*

/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\*

\* Паралельне програмування

\* Курсова робота. ПРГ1. Бібліотека OpenMP \*

\*

\* Завдання: A = max(Z)\*B + min(Z)\*alfa\*B\*(MO\*MK)

\*

\* Файл operations.cpp

\* Автор Кобилинський Дмитро

\* Група ІП-31

\* Дата 20.03.16

\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\*/

#include "stdafx.h"

#include <windows.h>

using namespace std;

#include "operations.h"

#include <iostream>

/\*Введення вектора\*/

vector inputVector(int value){

vector result = new int[N];

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

{

result[i] = value;

}

return result;

}

/\*введення матриці\*/

matrix inputMatrix(int value){

matrix result = new vector[N];

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

{

result[i] = new int[N];

}

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

{

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

{

result[i][j] = value;

}

}

return result;

}

/\*Виведення вектора\*/

void output(vector v) {

if (N <= 20) {

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

cout << v[i] << " ";

}

cout << endl;

}

}

/\*Копіювання вектора\*/

vector copyVector(vector v){

vector result = new int[N];

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

{

result[i] = v[i];

}

return result;

}

/\*Копіювання матриці\*/

matrix copyMatrix(matrix m){

matrix result = new vector[N];

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

{

result[i] = new int[N];

}

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

{

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

{

result[i][j] = m[i][j];

}

}

return result;

}

/\*Ініціалізація матриці\*/

matrix initMatrix(){

matrix result = new vector[N];

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

{

result[i] = new int[N];

}

return result;

}

*Operations.h*  
/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\*

\* Паралельне програмування

\* Курсова робота. ПРГ1. Бібліотека OpenMP \*

\*

\* Завдання: A = max(Z)\*B + min(Z)\*alfa\*B\*(MO\*MK)

\*

\* Файл operations.h

\* Автор Кобилинський Дмитро

\* Група ІП-31

\* Дата 20.03.16

\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\*/

typedef int\* vector;

typedef int\*\* matrix;

const int N = 4;

vector inputVector(int);

matrix inputMatrix(int);

void output(vector);

void output(matrix);

vector copyVector(vector);

matrix copyMatrix(matrix);

matrix initMatrix();

**Додаток Д. Структурна схема ПКС ЛП**

**Додаток Е. Схема алгоритму головної програми зі вказанням паралельних ділянок для ПРГ2**

**Додаток Ж. Схема алгоритму процесів для програми ПРГ2**

## Додаток И.

## Лістинг програми ПРГ2

Main.java

**import** mpi.Graphcomm;  
**import** mpi.MPI;  
  
**import** java.util.Arrays;  
  
*/\*\*  
 \* Course work PP-2 . MPI  
 \* Kobylynskiy D.A. IP-31   
 \*/***public class** Main {  
  
 **private static int**[][] *MK*, *MO*;  
 **private static int**[] *Z*,*B*,*A*;  
 **public static int** *a*, *minZ*,*maxZ*, *N*, *H*;  
 **private static int** *p*;  
  
 **public static void** main(String[] args) {  
  
 MPI.*Init*(args);  
  
 **long** startTime = System.*currentTimeMillis*();  
 **int** rank = MPI.*COMM\_WORLD*.Rank();  
 **int** size = MPI.*COMM\_WORLD*.Size();  
  
 System.***out***.println(**"Process "** + rank + **" is started"**);  
  
 *N* = Integer.*parseInt*(args[args.**length** - 1]);  
 *H* = *N* / size;  
 *p* = size - 1;  
  
 **int** [] dims = **new int**[2];  
 **boolean** [] periods = **new boolean**[2];  
 **for**(**int** i=0;i<dims.**length**;i++){  
 dims[i]=*N*;  
 }  
 **for**(**int** i=0;i<periods.**length**;i++){  
 periods[i]=**false**;  
 }  
 *//создаем топологию на основе глобального коммуникатора - каждый процесс будет знать своих соседей  
 // (необходимо для передачи данных), таким образом реализовываем решетку* Cartcomm cartcomm = MPI.*COMM\_WORLD*.Create\_cart(dims,periods,**false**);  
 *//ввод данных  
 initData*(rank);  
 **int**[] indexCount = *getEachCount*(size);  
 **int**[] displacement = *getEachOffset*(size);  
  
 **if** (rank == 0 || rank == *p*) {  
 *//устанавливаем метаданные для упаковки данных* DataPackBuilder.*setMetadata*(size, *getEachCount*(size), *getEachOffset*(size));  
 }  
  
 *//упаковываем данные для отправки* DataPack[] sendPacks;  
 **if** (rank == 0)  
 sendPacks = DataPackBuilder.*packData*(*MO*, *B*, *a*, **true**);  
 **else if** (rank == *p*)  
 sendPacks = DataPackBuilder.*packData*(*MK*, *Z*, 0, **false**);  
 **else** sendPacks = **new** DataPack[0];  
  
 *//указываем кол-во отсылаемых элементов каждому процессу (1 и последний процесс отсылают по пакету на процесс,  
 // остальные - не отсылают данные)* **int**[] sendCount = **new int**[size];  
  
 *//смещение в отсылаемом буффере для каждого процесса* **int**[] sendDisplacement = **new int**[size];  
  
 **if** (rank == 0 || rank == *p*) {  
  
 *//1 и последний процесс отсылают по пакету на процесс* Arrays.*fill*(sendCount, 1);  
 *//указываем смещение в пакетах (1 процесс начиная с 0 позиции получает 1 пакет, 2 - начиная с первой  
 // позиции получает 1 пакет и тд)* **for** (**int** i = 0; i < sendDisplacement.**length**; i++) {  
 sendDisplacement[i] = i;  
 }  
 }  
  
 **int**[] receiveCount = **new int**[size];  
 receiveCount[0] = 1;  
 receiveCount[*p*] = 1;  
  
 **int**[] receiveDisplacement = **new int**[size];  
 receiveDisplacement[0] = 0;  
 receiveDisplacement[*p*] = 1;  
  
 *// массив полученых пакетов* DataPack[] receivePacks = **new** DataPack[2];  
  
 cartcomm.Alltoallv(sendPacks, 0, sendCount, sendDisplacement, MPI.*OBJECT*,  
 receivePacks, 0, receiveCount, receiveDisplacement, MPI.*OBJECT*);  
  
 *//распаковка данных с 0 процесса  
 MO* = receivePacks[0].getMatrix2();  
 *a* = receivePacks[0].getConstant();  
 *B* = receivePacks[0].getVector();  
 *//распаковка данных с p процесса  
 MK* = receivePacks[1].getMatrix2();  
 *Z* = receivePacks[1].getVector();  
  
 **int**[] localMaximums = **new int**[1];**int**[] localMinimums = **new int**[1];  
 localMinimums[0] = MatrixOperations.*min*(*Z*, 0, *Z*.**length**);  
 localMaximums[0] = MatrixOperations.*max*(*Z*,0,*Z*.**length**);  
 *//вычисление глобального минимума и максимума* cartcomm.Reduce(localMinimums, 0, localMinimums, 0, 1, MPI.*INT*, MPI.*MIN*, *p*);  
 cartcomm.Bcast(localMinimums, 0, 1, MPI.*INT*, *p*);  
 cartcomm.Reduce(localMaximums, 0, localMaximums, 0, 1, MPI.*INT*, MPI.*MAX*, *p*);  
 cartcomm.Bcast(localMaximums, 0, 1, MPI.*INT*, *p*);  
 *minZ* = localMinimums[0];  
 *maxZ* = localMaximums[0];  
 **int** multConst = *minZ*\**a*;  
  *A* = MatrixOperations.*addVectors*(MatrixOperations.*multVectorOnConst*(*B*,*maxZ*),MatrixOperations.*multVectorOnConst*(MatrixOperations.*multVectorMatrix*(*B*,MatrixOperations.*multMatrix*(*MO*,*MK*)),multConst),1,1);  
 System.***out***.println(Arrays.*toString*(localMinimums));  
**int**[] result = **new int**[*N*];  
 cartcomm.Gatherv(*A*, 0, indexCount[rank], MPI.*OBJECT*, result, 0, indexCount, displacement, MPI.*OBJECT*, 0);  
 *//вывод результатов на экран и завершение работы* **if** (rank == 0) {  
 **if** (result.**length** <= 16)  
 System.***out***.println(MatrixOperations.*formattedDeepToString*(result));  
 System.***out***.println(**"Computation time (sec): "** + ((System.*currentTimeMillis*() - startTime) / 1000.0));  
 }  
 System.***out***.println(**"Process "** + rank + **" is finished"**);  
 MPI.*Finalize*();  
 }  
  
 **private static void** initData(**int** rank) {  
 **if** (rank == 0) {  
 *B* = MatrixOperations.*inputVector*(*N*);  
 *MO* = MatrixOperations.*inputMatrix*(*N*);  
 *a* = MatrixOperations.*inputConstant*();  
  
 } **else if** (rank == *p*) {  
 *MK* = MatrixOperations.*inputMatrix*(*N*);  
 *Z* = MatrixOperations.*inputVector*(*N*);  
 }  
 }  
  
 **int**[] getEachCount(**int** processCount) {  
 **int**[] result = **new int**[processCount];  
 Arrays.*fill*(result, 1, result.**length**, *H*);  
 *//первый процесс получит больше элементов, если их неравное количество на каждый процесс* result[0] = *N* - (result.**length** - 1) \* *H*;  
 **return** result;  
 }  
  
 **private static int**[] getEachOffset(**int** processCount) {  
 **int**[] result = **new int**[processCount];  
 result[0] = 0;  
 result[1] = *N* - (result.**length** - 1) \* *H*;  
 **for** (**int** i = 2; i < result.**length**; i++) {  
 result[i] = result[i - 1] + *H*;  
 }  
 **return** result;  
 }  
}

MatrixOperations.java

**import** java.util.Arrays;  
  
*/\*\*  
 \* Course work PP-2 . MPI  
 \* Kobylynskiy D.A. IP-31  
 \*/***public class** MatrixOperations {  
  
**public static int**[][] inputMatrix(**int** n) {  
 **int**[][] result = **new int**[n][n];  
 **for** (**int** i = 0; i < n; i++)  
 **for** (**int** j = 0; j < n; j++) {  
 result[i][j] = 1;  
 }  
 **return** result;  
 }  
  
 **public static int**[] inputVector(**int** n) {  
 **int**[] result = **new int**[n];  
 **int** start = 4;  
 **for** (**int** i = 0; i < result.**length**; i++) {  
 result[i] = start + i;  
 }  
 **return** result;  
 }  
  
 **public static int** inputConstant() {  
 **return** 1;  
 }  
  
**public static int**[][] addMatrix(**int**[][] param1, **int**[][] param2, **int** const1, **int** const2) {  
 **if** (param1.**length** != param2.**length**) {  
 System.***out***.println(**"Нельзя суммировать матрицы с разным количеством строк"**);  
 **return null**;  
 }  
 **int**[][] result = **new int**[param1.**length**][param1[0].**length**];  
 **for** (**int** i = 0; i < result.**length**; i++) {  
 result[i] = *addVectors*(param1[i], param2[i], const1, const2);  
 }  
 **return** result;  
 }  
  
 **public static int**[] addVectors(**int**[] p1, **int**[] p2, **int** const1, **int** const2) {  
 **int**[] result = **new int**[p1.**length**];  
 **for** (**int** i = 0; i < result.**length**; i++) {  
 result[i] = p1[i] \* const1 + p2[i] \* const2;  
 }  
 **return** result;  
 }  
  
 **public static int**[][] multMatrix(**int**[][] param1, **int**[][] param2) {  
 **int**[][] result = **new int**[param1.**length**][param1.**length**];  
 **for** (**int** k = 0; k < param1.**length**; k++) {  
 **for** (**int** i = 0; i < param1[0].**length**; i++) {  
 **for** (**int** j = 0; j < param2.**length**; j++) {  
 result[k][i] += param1[k][j] \* param2[j][i];  
 }  
 }  
 }  
 **return** result;  
 }  
  
 **public static int**[] multVectorMatrix(**int**[] param1,**int**[][] param2){  
 **if** (param1.**length** != param2.**length**) {  
 System.***out***.println(**"Нельзя умножать вектор на матрицу, количество элементов которого не равно кол-ву столбцов матрицы"**);  
 **return null**;  
 }  
 **int**[] result = **new int**[param1.**length**];  
 **for** (**int** k = 0; k < param1.**length**; k++) {  
 **for** (**int** i = 0; i < param1.**length**; i++) {  
 **for** (**int** j = 0; j < param2.**length**; j++) {  
 result[k] += param1[j] \* param2[j][i];  
 }  
 }  
 }  
 **return** result;  
 }  
  
 **public static int**[] multVectorOnConst(**int**[] param1,**int** constant){  
 **int**[] result = **new int**[param1.**length**];  
 **for** (**int** i=0; i<param1.**length**;i++){  
 result[i]=param1[i]\*constant;  
 }  
 **return** result;  
 }  
  
 **public static** String formattedDeepToString(**int**[] matrix) {  
 StringBuilder result = **new** StringBuilder();  
 result.append(Arrays.*toString*(matrix)).append(**"\n"**);  
 **return** result.toString();  
 }  
  
 **public static int** min(**int**[] vector, **int** startIndex, **int** endIndex) {  
 **int** result = 0;  
 **for** (**int** i = startIndex + 1; i < endIndex; i++) {  
 result = result > vector[i] ? vector[i] : result;  
 }  
 **return** result;  
 }  
 **public static int** max(**int**[] vector, **int** startIndex, **int** endIndex){  
 **int** result = 9999;  
 **for** (**int** i = startIndex + 1; i < endIndex; i++) {  
 result = result < vector[i] ? vector[i] : result;  
 }  
 **return** result;  
 }  
 **public static int**[][] truncateMatrix(**int**[][] matrixToTruncate, **int** offset, **int** rowCount) {  
 **int** result[][] = **new int**[rowCount][];  
 System.*arraycopy*(matrixToTruncate, offset, result, 0, rowCount);  
 **return** result;  
 }  
  
}

DataPack.java

**import** java.io.Serializable;  
  
*/\*\*  
 \* Course work PP-2 . MPI  
 \* Kobylynskiy D.A. IP-31   
 \*/***public class** DataPack **implements** Serializable {  
  
 **private int**[][] **matrix1**;  
  
 **private int**[] **vector**;  
  
 **public** DataPack() {  
 **matrix1** = **new int**[0][0];  
 **vector** = **new int**[0];  
 }  
  
 **public int** getConstant() {  
 **return constant**;  
 }  
  
 **public void** setConstant(**int** constant) {  
 **this**.**constant** = constant;  
 }  
  
 **public int**[] getVector() {  
 **return vector**;  
 }  
  
 **public void** setVector(**int**[] vector) {  
 **this**.**vector** = vector;  
 }  
   
  
 **public int**[][] getMatrix1() {  
 **return matrix1**;  
 }  
  
 **public void** setMatrix1(**int**[][] matrix1) {  
 **this**.**matrix1** = matrix1;  
 }  
  
 **private int constant**;  
  
}

DataPackBuilder.java

**import** java.util.Arrays;  
  
*/\*\*  
 \* Course work PP-2 . MPI  
 \* Kobylynskiy D.A. IP-31  
 \*/***public class** DataPackBuilder {  
  
 **private static** DataPack[] *preparedPacks*;  
  
 **public static int**[] *elementsCount*;  
  
 **public static int**[] *dataOffset*;  
  
 **public static void** setMetadata(**int** processCount, **int**[] elemCount, **int**[] offset) {  
 *preparedPacks* = **new** DataPack[processCount];  
 **for** (**int** i = 0; i < *preparedPacks*.**length**; i++) {  
 *preparedPacks*[i] = **new** DataPack();  
 }  
 *elementsCount* = elemCount;  
 *dataOffset* = offset;  
 }  
  
 **public static** DataPack[] packData(**int**[][] matrix1, **int**[] vector, **int** constant, **boolean** matrix1FullSize) {  
 **for** (**int** i = 0; i < *preparedPacks*.**length**; i++) {  
 **if** (matrix1FullSize)  
 *preparedPacks*[i].setMatrix1(matrix1);  
 **else** *preparedPacks*[i].setMatrix1(Arrays.*copyOfRange*(matrix1, *dataOffset*[i], *dataOffset*[i] + *elementsCount*[i]));  
 **if** (vector != **null**)  
 *preparedPacks*[i].setVector(Arrays.*copyOfRange*(vector, *dataOffset*[i], *dataOffset*[i] + *elementsCount*[i]));  
 *preparedPacks*[i].setConstant(constant);  
 }  
 **return** *preparedPacks*;  
 }  
}