Національний 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 ІО-01

напряму підготовки 050102

«Комп’ютерна інженерія»

Чеканський Є. 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 – 2013 рік

Національний технічний університет України  
«Київський політехнічний інститут»

Факультет (інститут) інформатики та обчислювальної техніки

(повна назва)

Кафедра обчислювальної техніки

(повна назва)

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.050102 «Комп’ютерна інженерія»

(шифр і назва)

**Завдання**

на курсову роботу студенту

Чеканський Євгеній 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 11 травня 2013 р.
2. Вихідні дані для роботи:

* огляд засобів роботи з процесами в бібліотеці PVM;
* математична задача ;
* структури ПКС СП та ПКС ЛП;
* бібліотеки програмування: Java, Ada Рандеву.

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 |
|  | Виконання огляду для розділу 1 | 20.03.2013 |
|  | Розробка паралельного алгоритму рішення задачі | 01.04.2013 |
|  | Розробка алгоритмів процесів | 06.04.2013 |
|  | Розробка схем взаємодії процесів | 13.04.2013 |
|  | Розробка програм | 20.04.2013 |
|  | Тестування програм | 30.04.2013 |
|  | Оформлення КР | 10.05.2013 |
|  | Захист КР | 18.05.2013 |

Студент

(підпис) (прізвище та ініціали)

Керівник роботи

(підпис) (прізвище та ініціали)

ПОЯСНЮВАЛЬНА ЗАПИСКА

ЗМІСТ

[ВСТУП 7](#_Toc356248616)

[РОЗДІЛ 1. ОГЛЯД ЗАСОБІВ РОБОТИ З ПРОЦЕСАМИ В БІБЛІОТЕЦІ PVM 8](#_Toc356248617)

[1.1 Загальна інформація 8](#_Toc356248618)

[1.2 Загальна структура простої програми 11](#_Toc356248619)

[1.3 Створення нової задачі 11](#_Toc356248620)

[1.2 Передача повідомлень 13](#_Toc356248621)

[1.3 Робота з буферами 13](#_Toc356248622)

[1.4 Пакування і розпакування повідомлень 15](#_Toc356248623)

[1.5 Передавання і приймання повідомлень 15](#_Toc356248624)

[1.6 Робота з групами 17](#_Toc356248625)

[1.7 Приклад використання 18](#_Toc356248626)

[1.8 Висновки по розділу 1 18](#_Toc356248629)

[РОЗДІЛ 2. Розробка програми ПРГ1 для ПКС СП 19](#_Toc356248630)

[2.1 Розробка паралельного математичного алгоритму 19](#_Toc356248631)

[2.2 Розробка алгоритмів процесів 20](#_Toc356248634)

[2.3 Розробка схеми взаємодії процесів 22](#_Toc356248637)

[2.4 Розробка програми ПРГ1 23](#_Toc356248638)

[2.5 Тестування програми ПРГ1 23](#_Toc356248639)

[2.6 Висновки до розділу 2 26](#_Toc356248640)

[РОЗДІЛ 3. Розробка програми ПРГ2 для ПКС ЛП 27](#_Toc356248641)

[3.1 Розробка паралельного математичного алгоритму 27](#_Toc356248642)

[3.2 Розробка алгоритмів процесів 27](#_Toc356248643)

[3.3 Розробка схеми взаємодії процесів 29](#_Toc356248646)

[3.4 Розробка програми ПРГ2 30](#_Toc356248647)

[3.5 Тестування програми ПРГ2 30](#_Toc356248648)

[3.6 Висновки до розділу 3 33](#_Toc356248649)

[ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ПО РОБОТІ 34](#_Toc356248650)

[СПИСОК ЛІТЕРАТУРИ 35](#_Toc356248651)

[ДОДАТКИ 36](#_Toc356248652)

[Додаток А. Структурна схема ПКС СП 37](#_Toc356248653)

[Додаток Б. Схема алгоритму головної програми зі вказанням паралельних ділянок для ПРГ1 39](#_Toc356248654)

[Додаток В. Схеми алгоритмів процесів для програми ПРГ1 41](#_Toc356248655)

[Додаток Г. Лістинг приграми ПРГ1 48](#_Toc356248656)

[Додаток Д. Структурна схема ПКС ЛП 59](#_Toc356248657)

[Додаток Е. Схема алгоритму головної програми зі вказанням паралельних ділянок для ПРГ2 61](#_Toc356248658)

[Додаток Є. Схеми алгоритмів задач для програми ПРГ2 63](#_Toc356248659)

[Додаток Ж. Лістинг програми ПРГ2 70](#_Toc356248660)

# ВСТУП

Курсова робота по дисципліні «Паралельні і розподілені обчислення» складається з трьох розділів.

В першому розділі «Огляд засобів роботи з процесами в бібліотеці PVM» Описані основні принципи роботи бібліотеки PVM і наведено приклади найголовніших методів.

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

Лістинги та алгоритми розроблених програм наведено у додатках.

# РОЗДІЛ 1. ОГЛЯД ЗАСОБІВ РОБОТИ З ПРОЦЕСАМИ В БІБЛІОТЕЦІ 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

## 1.5 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

## 1.6 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

## 1.7 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.

## 1.8 Висновки по розділу 1

Виконано загальний огляд бібліотеки PVM. Було 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. Було 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 ПРГ1 для ПКС СП

## 2.1 Розробка паралельного математичного алгоритму

Вираз математичного завдання може бути обчислений в 4 етапи для 6 задач:

де i = 1..6

де j=1,3,5

Загальні ресурси : B, C, MR

sort – сортування бульбашкою, sortm – сортування злиттям.

Така кількість етапів пов’язана з сортуванням. Кількість етапів буде збільшуватись з кількістю задач(перші 8 задач – 4 етапи, кожних додаткових 8 задач – 1 додатковий етап).

Оцінимо мінімальний час обчислення функції згідно з концепцією необмеженого паралелізму.

Розглянемо операції, що використані в математичній функції, і їхній час виконання:

* Множення матриць -
* Множення вектора на матрицю -
* Додавання векторів, сортування злиттям -

## Час виконання різних етапів обчислення наведено в таблиці 2.1.

Таблиця 2.1. Час виконання різних етапів

|  |  |
| --- | --- |
| Етап | Час |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Час виконання задачі в системах з різною кількістю процесорів наведений в таблиці 2.2.

Таблиця 2.2. Час виконання задачі в системах з різною кількістю процесорів

|  |  |
| --- | --- |
| Кількість процесорів | Час виконання |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |

Очевидно, що коефіцієнти прискорення і коефіцієнти ефективності будуть максимальними.

## 2.2 Розробка алгоритмів процесів

Задача Т1:

|  |  |
| --- | --- |
| 1. Очікування вводу Т2,Т4,Т6 | W2,4,6-1,2,3 |
| 1. Копіювання B2 = B, С2=С | КД |
| 1. Копіювання MR2 = MR | КД |
| 1. Рахунок |  |
| 1. Сигнал Т2 про завершення рахунку | S2-4 |

Задача Т2:

|  |  |
| --- | --- |
| 1. Введення B,C |  |
| 1. Сигнал задачам Т1,Т3-Т6 про завершення вводу | S1,3..6-1 |
| 1. Очікування вводу задач Т4, Т6 | W4,6-2,3 |
| 1. Копіювання B2 = B, С2=С | КД |
| 1. Копіювання MR2 = MR | КД |
| 1. Рахунок |  |
| 1. Очікування рахунку Т1 | W1-4 |
| 1. Рахунок |  |
| 1. Очікування рахунку Т3 | W3-8 |
| 1. Рахунок |  |
| 1. Вивід А |  |

Задача Т3:

|  |  |
| --- | --- |
| 1. Очікування вводу Т2,Т4,Т6 | W2,4,6-1,2,3 |
| 1. Копіювання B2 = B, С2=С | КД |
| 1. Копіювання MR2 = MR | КД |
| 1. Рахунок |  |
| 1. Очікування рахунку Т4 | W4-6 |
| 1. Рахунок |  |
| 1. Очікування рахунку Т5 | W5-7 |
| 1. Рахунок |  |
| 1. Сигнал задачі Т2 про завершення рахунку | S2-8 |

Задача Т4:

|  |  |
| --- | --- |
| 1. Введення MO, MT |  |
| 1. Сигнал задачам Т1-Т3, Т5, Т6 про завершення вводу | S1..3,5..6-2 |
| 1. Очікування вводу Т2, Т6 | W2,6-1,3 |
| 1. Копіювання B2 = B, С2=С |  |
| 1. Копіювання MR2 = MR |  |
| 1. Рахунок |  |
| 1. Сигнал задачі Т3 про завершення рахунку | S3-6 |

Задача Т5:

|  |  |
| --- | --- |
| 1. Очікування вводу Т2,Т4,Т6 | W2,4,6-1,2,3 |
| 1. Копіювання B2 = B, С2=С | КД |
| 1. Копіювання MR2 = MR | КД |
| 1. Рахунок |  |
| 1. Очікування рахунку Т6 | W6-5 |
| 1. Рахунок |  |
| 1. Сигнал задачі Т3 про завершення рахунку | S3-7 |

Задача Т6:

|  |  |
| --- | --- |
| 1. Введення MR |  |
| 1. Сигнал задачам Т1 - Т5 про завершення вводу | S1..5-3 |
| 1. Очікування вводу Т2,Т4 | W2,4-1,2 |
| 1. Копіювання B2 = B, С2=С | КД |
| 1. Копіювання MR2 = MR | КД |
| 1. Рахунок |  |
| 1. Сигнал задачі Т5 про завершення рахунку | S5-5 |

|  |  |  |
| --- | --- | --- |
| Рис.2.1. Структурна схема взаємодії процесів | Структурна схема взаємодії процесів зображена на рис. 2.1. | 2.3 Розробка схеми взаємодії процесів |

## 2.4 Розробка програми ПРГ1

ПЗ для ПКС зі спільною пам'яттю реалізоване на мові програмування Java[2] з використанням засобів паралельного програмування стандартної бібліотеки Java та складається з наступних файлів:

1. Cw.java – містить опис задач та математичної функції, точку входу в програму
2. Operations.java – містить клас опису математичних операцій над векторами і матрицями

Основні змінні класу Operations:

1. N – розмір даних
2. P – кількість потоків

Основні методи класу Operations:

1. vector\_matrix\_mul\_P – множення вектору на матрицю
2. matrix\_mul\_P – множення матриці на матрицю
3. vector\_add\_P – додавання вектора до вектора
4. merge\_sort – сортування злиттям

Основні змінні класу Cw:

1. s\_inputT2,4,6 – семафори для синхронізації по введенню
2. s\_sort\_0\_T1,6,4 – семафори для синхронізації першого сортування
3. s\_sort\_1\_T5 – семафор для синхронізації другого сортування
4. s\_sort\_2\_T3 – семафор для синхронізації останнього сортування
5. cs\_copyB\_C – об’єкт для синхронізованого блоку(копіювання В,С)
6. cs\_copyMR – об’єкт для синхронізованого блоку (копіювання MR)

Також в класі Cw містяться внутрішні класи для опису кожної з задач(Task1-6) і змінні для зберігання результатів проміжних сортувань.

Лістинг програми наведений в Додатку.

## 2.5 Тестування програми ПРГ1

Значення часу Т1-Т4 для різних N відображаються в таблиці 2.3.

Таблиця 2.3. Час виконання програми в ПКС зі СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N / P | 1 | 2 | 3 | 4 |
| 900 | 20113 | 10123 | 6800 | 5282 |
| 1800 | 158370 | 81548 | 53968 | 41603 |
| 2400 | 375195 | 190918 | 128495 | 95425 |

На підставі значень часу Т1-Т4 виконуються розрахунки коефіцієнтів прискорення Кпn = Т1/Тn, які відображаються в таблиці 2.4.

Таблиця 2.4. Значення коефіцієнту прискорення для ПКС зі СП

|  |  |  |  |
| --- | --- | --- | --- |
| N / P | 2 | 3 | 4 |
| 900 | 1,986861 | 2,957794 | 3,807837 |
| 1800 | 1,942046 | 2,934516 | 3,806696 |
| 2400 | 1,965215 | 2,919919 | 3,931831 |

Графіки зміни Кп в залежності від значень Р = (1 - 4) для різних N зображені на рис. 2.2 - рис. 2.4.

Рис. 2.2. Залежність коефіцієнту прискорення від кількості ядер для ПКС з спільною пам’яттю при N = 900

Рис. 2.3. Залежність коефіцієнту прискорення від кількості ядер для ПКС з спільною пам’яттю при N = 1800

Рис. 2.4. Залежність коефіцієнту прискорення від кількості ядер для ПКС з спільною пам’яттю при N = 2400

Виконаємо розрахунки коефіцієнтів ефективності Кеn = Тn / Р \* 100%, які відображаються у таблиці 2.5.

Таблиця 2.5.Значення коефіцієнту ефективності для ПКС з СП

|  |  |  |  |
| --- | --- | --- | --- |
| N / P | 2 | 3 | 4 |
| 900 | 99,3431 | 98,5931 | 95,1959 |
| 1800 | 97,1023 | 97,8172 | 95,1679 |
| 2400 | 98,2608 | 97,3306 | 98,2958 |

Графіки зміни Ке в залежності від значень Р = (1 - 4) для різних N зображені на рис. 2.5.

Рис. 2.5. Залежність коефіцієнту ефективності від кількості ядер для ПКС з спільною пам’яттю

## 2.6 Висновки до розділу 2

Виконано розробку програми ПРГ1 для ПКС СП з використанням мови Java і засобів синхронізації стандартної бібліотеки Java. Тестування програми показало наступне:

* використання багатоядерної ПКС та програми ПРГ1 забезпечує скорочення часу обчислення заданої математичної функції. Значення Кп змінюється від 1,942046 до 3,931831
* максимальне значення Кп забезпечує ПКС з Р=4 та N=2400
* мінімальне значення Кп забезпечує ПКС з Р=2 та N=1800
* з ростом N Кп з не великим відхиленням зростає на 1

Нерівномірна зміна коефіцієнту ефективності пояснюється роботою GC(garbage collector[3]). Під час декількох запусків програми з незмінними параметрами, Ке змінювався в невеликих межах. Це пов’язано з тим, що середовище Java автоматично звільняє використані ресурси, але момент часу звільнення не можливо передбачити, тому звільнення ресурсів може відбуватися під час розрахунків, відбираючи процесорний час у задач.

# РОЗДІЛ 3. Розробка програми ПРГ2 для ПКС ЛП

## 3.1 Розробка паралельного математичного алгоритму

Оскільки математичний вираз співпадає з тим, що був розглянутий в розділі 2.1, то перший крок не матиме відмінностей від уже виконаного(згідно з рекомендованою методикою[4]).

## 3.2 Розробка алгоритмів процесів

Задача Т1:

1. Введення B,C
2. **Передати** задачі Т2 B,C
3. **Передати** задачі Т6 B,C
4. **Прийняти** з задачі Т2
5. **Прийняти** з задачі Т6
6. Обчислення
7. **Прийняти** з задачі Т2
8. Обчислення
9. **Прийняти** з задачі Т6
10. Обчислення
11. Вивід А

Задача Т2:

1. **Прийняти** з задачі Т1 B,C
2. **Передати** задачі Т3 B,C
3. **Прийняти** з задачі Т3
4. **Передати** задачі Т1
5. **Прийняти** з задачі Т3
6. Обчислення
7. **Передати** задачі Т1

Задача Т3:

1. Введення MO, MT
2. **Прийняти** з задачі Т2 B,C
3. **Передати** задачі Т2
4. **Передати** задачі Т4
5. **Прийняти** з задачі Т4
6. **Передати** задачі Т2
7. Обчислення
8. **Передати** задачі Т4

Задача Т4:

1. **Прийняти** з задачі Т5 B,C
2. **Прийняти** з задачі Т3
3. **Передати** задачі Т5
4. **Прийняти** з задачі Т5
5. **Передати** задачі Т3
6. Обчислення
7. **Прийняти** з задачі Т3
8. Обчислення
9. **Передати** задачі Т5

Задача Т5:

1. Введення MR
2. **Прийняти** з задачі Т6 B,C
3. **Передати** задачі Т4 B,C
4. **Прийняти** з задачі Т4
5. **Передати** задачі Т6
6. **Передати** задачі Т6
7. **Передати** задачі Т4
8. Обчислення
9. **Передати** задачі Т6
10. **Прийняти** з задачі Т4
11. **Передати** задачі Т6

Задача Т6:

1. **Прийняти** з задачі Т1 B,C
2. **Передати** задачі Т5 B,C
3. **Прийняти** з задачі Т5
4. **Прийняти** з задачі Т5
5. **Передати** задачі Т1
6. Обчислення
7. **Прийняти** з задачі Т5
8. Обчислення
9. **Прийняти** з задачі Т5
10. Обчислення
11. **Передати** задачі Т1

|  |  |  |
| --- | --- | --- |
| Рис.3.1. Структурна схема взаємодії процесів | Структурна схема взаємодії процесів зображена на рис. 3.1. | 3.3 Розробка схеми взаємодії процесів |

## 3.4 Розробка програми ПРГ2

ПЗ для ПКС з розподіленою пам'яттю реалізоване на мові програмування Ada з використанням засобів паралельного програмування Ada-Рандеву[5], та складається з наступних файлів:

1. data.ads – містить опис пакету математичних і допоміжних функцій
2. data.ads – містить тіло пакету математичних і допоміжних функцій
3. main.adb – містить опис задач, точку входу в програму (процедура Main)

Пакет data не містить в собі параметрів, всі необхідні параметри зберігаються в 0 елементі вектору чи матриці. Матриця реалізована через масив цілих чисел, де 0 елемент – розмір матриці, наступні елементи – стовбці матриці.

Основні методи пакету data:

1. Vector\_Matrix\_Mul\_P – множення вектору на матрицю
2. Matrix\_Mul – множення матриці на матрицю
3. Vector\_Add – додавання вектора до вектора
4. VecMergeSort – сортування злиттям
5. VecRangeSort – сортування «бульбашкою»[]
6. VecCRange, MVecCRange, VecWRange, MVecWRange – процедури для зчитування/запису певної частини вектора/матриці

Методи VecCRange, MVecCRange, VecWRange, MVecWRange використовуються для передачі лише необхідної частини матриці/вектора, для реалізації ефективного обміну даними в системі з розподіленою пам’яттю.

Основні змінні процедури Main:

* N – розмір векторів і матриць
* PC – кількість задач

Також в процедурі Main за допомогою блоку declare описані задачі Т1-6 з відповідними входами(entry) для правильної роботи Ada-Рандеву. Лістинг програми наведений в Додатку.

## 3.5 Тестування програми ПРГ2

Значення часу Т1-Т4 для різних N відображаються в таблиці 3.1.

Таблиця 3.1. Час виконання програми в ПКС зі ЛП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N / P | 1 | 2 | 3 | 4 |
| 900 | 22315 | 11218 | 7502 | 5742 |
| 1800 | 179122 | 93123 | 61097 | 45233 |
| 2400 | 424313 | 230142 | 146479 | 107767 |

На підставі значень часу Т1-Т4 виконуються розрахунки коефіцієнтів прискорення Кпn = Т1/Тn, які відображаються в таблиці 3.2.

Таблиця 3.2. Значення коефіцієнту прискорення для ПКС з ЛП

|  |  |  |  |
| --- | --- | --- | --- |
| N / P | 2 | 3 | 4 |
| 900 | 1,989213 | 2,974540 | 3,886276 |
| 1800 | 1,923499 | 2,931764 | 3,959984 |
| 2400 | 1,843701 | 2,896749 | 3,937318 |

Графіки зміни Кп в залежності від значень Р = (1 - 4) для різних N зображені на рис. 3.2 - рис. 3.4.

Рис. 3.2. Залежність коефіцієнту прискорення від кількості ядер для ПКС з ЛП при N = 900

Рис. 3.3. Залежність коефіцієнту прискорення від кількості ядер для ПКС з ЛП при N = 1800

Рис. 3.4. Залежність коефіцієнту прискорення від кількості ядер для ПКС з ЛП при N = 2400

Виконаємо розрахунки коефіцієнтів ефективності Кеn = Тn / Р \* 100%, які відображаються у таблиці 3.3.

Таблиця 3.3.Значення коефіцієнту ефективності для ПКС з ЛП

|  |  |  |  |
| --- | --- | --- | --- |
| N / P | 2 | 3 | 4 |
| 900 | 99,4607 | 99,1513 | 97,1569 |
| 1800 | 96,1749 | 97,7255 | 98,9998 |
| 2400 | 92,1851 | 96,5583 | 98,4329 |

Графіки зміни Ке в залежності від значень Р = (1 - 4) для різних N зображені на рис. 3.5.

Рис. 3.5. Залежність коефіцієнту ефективності від кількості ядер для ПКС з ЛП

## 3.6 Висновки до розділу 3

Виконано розробку програми ПРГ2 для ПКС з ЛП з використанням мови Ada і засобу взаємодії задач Ada-Рандеву. Тестування програми показало наступне:

* використання багатоядерної ПКС та програми ПРГ1 забезпечує скорочення часу обчислення заданої математичної функції. Значення Кп змінюється від 1,843701 до 3,959984
* максимальне значення Кп забезпечує ПКС з Р=4 та N=1800
* мінімальне значення Кп забезпечує ПКС з Р=4 та N=2400
* з ростом N Кп з не великим відхиленням зростає на 1

# ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ПО РОБОТІ

* Бібліотека PVM стала побічним результатом досліджень гетерогенних розрахунків
* Бібліотека PVM дозволяє обчислювати задачі на різноманітних за архітектурою обчислювальних машинах
* За принципами роботи PVM схожа на MPI і основний спосіб взаємодії задач пов'язаний з пересилкою повідомлень
* З точки зору концепції необмеженого паралелізму, на системі з необмеженої кількістю процесорів додавання векторів і сортування злиттям виконуються за однаковий час
* Зменшення Кп в ПРГ1 та ПРГ2 пояснюється недостатньою пропускною здатністю пам’яті
* ПРГ1 працює швидше за ПРГ2. Це пояснюється тим, що в ПРГ2 передача даних відбувається через проміжні вузли системи з розподіленою пам’яттю
* Алгоритм ПРГ2 значно складніший за алгоритм ПРГ1 через складну топологію ПКС з РП. Велика кількість проміжних передач даних ускладнює реалізацію програми і збільшує вірогідність помилки

# 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. Java SE Documentation at a Glance [Electronic resource]. – Access mode: <http://www.oracle.com/technetwork/java/javase/documentation/index.html>. – Last access: 6.03.2013. – Title from the screen.
3. Java HotSpot Garbage Collection [Electronic resource]. – Access mode: <http://www.oracle.com/technetwork/java/javase/tech/gc-142724.html>. – Last access: 6.04.2013. – Title from the screen.
4. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення: Навч. посібник [Текст]. – К.: Корнійчук, 2005. – 226 с. – ISBN 996-7599-36-1.
5. The Rendezvous [Electronic resource]. – Access mode: <http://www2.adacore.com/gap-static/GNAT_Book/html/node22.htm>. – Last access: 6.04.2013. – Title from the screen.

# ДОДАТКИ

## Додаток А. Структурна схема ПКС СП

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

## Додаток В. Схеми алгоритмів процесів для програми ПРГ1

## Додаток Г. Лістинг приграми ПРГ1

**1. /\*--------------------------------------------**

**2. -- Cw.java --**

**3. -- A = sort(B\*MO+C\*(MT\*MR) --**

**4. -- Chekanskiy Evgeniy Anatoliyovich, IO-01 --**

**5. -- 5.04.2013 --**

**6. --------------------------------------------\*/**

**7. package crs.cw;**

**8.**

**9. import java.util.Arrays;**

**10. import java.util.Scanner;**

**11. import java.util.concurrent.Semaphore;**

**12.**

**13. public class Cw {**

**14. //commons**

**15. static int N;**

**16. static int P = 6;**

**17. static int PS\_1,PS\_2,PS\_3,PS\_4,PS\_5,PS\_6;**

**18. static Operations op;**

**19. //calc variables**

**20. static int[] A,B,C;**

**21. static int[][] MO,MT,MR;**

**22. static int[] T1\_sort\_0,T2\_sort\_0,T3\_sort\_0,T4\_sort\_0,T5\_sort\_0,T6\_sort\_0;**

**23. static int[] T2\_sort\_1,T3\_sort\_1,T5\_sort\_1;**

**24. static int[] T3\_sort\_2;**

**25. static int[] T2\_sort\_final;**

**26. //sync**

**27. static Semaphore inputT2 = new Semaphore(5);**

**28. static Semaphore inputT4 = new Semaphore(5);**

**29. static Semaphore inputT6 = new Semaphore(5);**

**30. static Semaphore sort\_0\_T1 = new Semaphore(1);**

**31. static Semaphore sort\_0\_T6 = new Semaphore(1);**

**32. static Semaphore sort\_0\_T4 = new Semaphore(1);**

**33. static Semaphore sort\_1\_T5 = new Semaphore(1);**

**34. static Semaphore sort\_2\_T3 = new Semaphore(1);**

**35. static Object copyB\_C = new Object();**

**36. static Object copyMR = new Object();**

**37. //static Object copyMR = new Object();**

**38. static class Task1 implements Runnable**

**39. {**

**40. int[] Bc,Cc;**

**41. int [][] MRc;**

**42. int part = 0;**

**43. @Override**

**44. public void run() {**

**45. try {**

**46.**

**47. inputT2.acquire();**

**48. inputT4.acquire();**

**49. inputT6.acquire();**

**50. synchronized(copyB\_C)**

**51. {**

**52. Bc = Arrays.copyOf(B,B.length);**

**53. Cc = Arrays.copyOf(C,C.length);**

**54. }**

**55. synchronized(copyMR)**

**56. {**

**57. MRc = op.matrix\_copy(MR);**

**58. }**

**59. int[] A\_TMP = op.vector\_add\_P(op.vector\_matrix\_mul\_P(Bc, MO, part), op.vector\_matrix\_mul\_P(Cc, op.matrix\_mul\_P(MT, MRc, part), part), part);**

**60. T1\_sort\_0 = op.vector\_copy\_P(A\_TMP, part);**

**61. Arrays.sort(T1\_sort\_0);**

**62. sort\_0\_T1.release();**

**63. } catch (InterruptedException e) {**

**64. }**

**65. }**

**66. }**

**67. static class Task2 implements Runnable**

**68. {**

**69. int[] Bc,Cc;**

**70. int [][] MRc;**

**71. int part = 1;**

**72. @Override**

**73. public void run() {**

**74. try {**

**75. B = op.vector\_create(1);**

**76. C = op.vector\_create(1);**

**77. inputT2.release(5);**

**78. inputT4.acquire();**

**79. inputT6.acquire();**

**80. synchronized(copyB\_C)**

**81. {**

**82. Bc = Arrays.copyOf(B,B.length);**

**83. Cc = Arrays.copyOf(C,C.length);**

**84. }**

**85. synchronized(copyMR)**

**86. {**

**87. MRc = op.matrix\_copy(MR);**

**88. }**

**89. int[] A\_TMP = op.vector\_add\_P(op.vector\_matrix\_mul\_P(Bc, MO, part), op.vector\_matrix\_mul\_P(Cc, op.matrix\_mul\_P(MT, MRc, part), part), part);**

**90. T2\_sort\_0 = op.vector\_copy\_P(A\_TMP, part);**

**91. Arrays.sort(T2\_sort\_0);**

**92. sort\_0\_T1.acquire();**

**93. op.merge\_sort(T1\_sort\_0, T2\_sort\_0, T2\_sort\_1, PS\_1, PS\_2);**

**94. sort\_2\_T3.acquire();**

**95. op.merge\_sort(T2\_sort\_1, T3\_sort\_2, T2\_sort\_final, PS\_1+PS\_2, PS\_3+PS\_4+PS\_5+PS\_6);**

**96. op.vector\_print(T2\_sort\_final);**

**97. } catch (InterruptedException e) {**

**98. }**

**99. }**

**100. }**

**101. static class Task3 implements Runnable**

**102. {**

**103. int[] Bc,Cc;**

**104. int [][] MRc;**

**105. int part = 2;**

**106. @Override**

**107. public void run() {**

**108. try {**

**109. inputT2.acquire();**

**110. inputT4.acquire();**

**111. inputT6.acquire();**

**112. synchronized(copyB\_C)**

**113. {**

**114. Bc = Arrays.copyOf(B,B.length);**

**115. Cc = Arrays.copyOf(C,C.length);**

**116. }**

**117. synchronized(copyMR)**

**118. {**

**119. MRc = op.matrix\_copy(MR);**

**120. }**

**121. int[] A\_TMP = op.vector\_add\_P(op.vector\_matrix\_mul\_P(Bc, MO, part), op.vector\_matrix\_mul\_P(Cc, op.matrix\_mul\_P(MT, MRc, part), part), part);**

**122. T3\_sort\_0 = op.vector\_copy\_P(A\_TMP, part);**

**123. Arrays.sort(T3\_sort\_0);**

**124. sort\_0\_T4.acquire();**

**125. op.merge\_sort(T3\_sort\_0, T4\_sort\_0, T3\_sort\_1, PS\_3, PS\_4);**

**126. sort\_1\_T5.acquire();**

**127. op.merge\_sort(T3\_sort\_1, T5\_sort\_1, T3\_sort\_2, PS\_3+PS\_4, PS\_5+PS\_6);**

**128. sort\_2\_T3.release();**

**129. } catch (InterruptedException e) {**

**130. }**

**131. }**

**132. }**

**133. static class Task4 implements Runnable**

**134. {**

**135. int[] Bc,Cc;**

**136. int [][] MRc;**

**137. int part = 3;**

**138. @Override**

**139. public void run() {**

**140. try {**

**141. MO = op.matrix\_create(1);**

**142. MT = op.matrix\_create(1);**

**143. inputT4.release(5);**

**144. inputT2.acquire();**

**145. inputT6.acquire();**

**146. synchronized(copyB\_C)**

**147. {**

**148. Bc = Arrays.copyOf(B,B.length);**

**149. Cc = Arrays.copyOf(C,C.length);**

**150. }**

**151. synchronized(copyMR)**

**152. {**

**153. MRc = op.matrix\_copy(MR);**

**154. }**

**155. int[] A\_TMP = op.vector\_add\_P(op.vector\_matrix\_mul\_P(Bc, MO, part), op.vector\_matrix\_mul\_P(Cc, op.matrix\_mul\_P(MT, MRc, part), part), part);**

**156. T4\_sort\_0 = op.vector\_copy\_P(A\_TMP, part);**

**157. Arrays.sort(T4\_sort\_0);**

**158. sort\_0\_T4.release();**

**159. } catch (InterruptedException e) {**

**160. }**

**161. }**

**162. }**

**163. static class Task5 implements Runnable**

**164. {**

**165. int[] Bc,Cc;**

**166. int [][] MRc;**

**167. int part = 4;**

**168. @Override**

**169. public void run() {**

**170. try {**

**171. inputT2.acquire();**

**172. inputT4.acquire();**

**173. inputT6.acquire();**

**174. synchronized(copyB\_C)**

**175. {**

**176. Bc = Arrays.copyOf(B,B.length);**

**177. Cc = Arrays.copyOf(C,C.length);**

**178. }**

**179. synchronized(copyMR)**

**180. {**

**181. MRc = op.matrix\_copy(MR);**

**182. }**

**183. int[] A\_TMP = op.vector\_add\_P(op.vector\_matrix\_mul\_P(Bc, MO, part), op.vector\_matrix\_mul\_P(Cc, op.matrix\_mul\_P(MT, MRc, part), part), part);**

**184. T5\_sort\_0 = op.vector\_copy\_P(A\_TMP, part);**

**185. Arrays.sort(T5\_sort\_0);**

**186. sort\_0\_T6.acquire();**

**187. op.merge\_sort(T5\_sort\_0, T6\_sort\_0, T5\_sort\_1, PS\_5, PS\_6);**

**188. sort\_1\_T5.release();**

**189. } catch (InterruptedException e) {**

**190. }**

**191. }**

**192. }**

**193. static class Task6 implements Runnable**

**194. {**

**195. int[] Bc,Cc;**

**196. int [][] MRc;**

**197. int part = 5;**

**198. @Override**

**199. public void run() {**

**200. try {**

**201. MR = op.matrix\_create(1);**

**202. inputT6.release(5);**

**203. inputT2.acquire();**

**204. inputT4.acquire();**

**205. synchronized(copyB\_C)**

**206. {**

**207. Bc = Arrays.copyOf(B,B.length);**

**208. Cc = Arrays.copyOf(C,C.length);**

**209. }**

**210. synchronized(copyMR)**

**211. {**

**212. MRc = op.matrix\_copy(MR);**

**213. }**

**214. int[] A\_TMP = op.vector\_add\_P(op.vector\_matrix\_mul\_P(Bc, MO, part), op.vector\_matrix\_mul\_P(Cc, op.matrix\_mul\_P(MT, MRc, part), part), part);**

**215. T6\_sort\_0 = op.vector\_copy\_P(A\_TMP, part);**

**216. Arrays.sort(T6\_sort\_0);**

**217. sort\_0\_T6.release();**

**218. } catch (InterruptedException e) {**

**219. }**

**220. }**

**221. }**

**222. public static void main(String[] args) throws InterruptedException {**

**223. Scanner s = new Scanner(System.in);**

**224. System.out.println("Enter N:");**

**225. N = s.nextInt();**

**226. inputT2.acquire(5);**

**227. inputT4.acquire(5);**

**228. inputT6.acquire(5);**

**229. sort\_0\_T1.acquire();**

**230. sort\_0\_T4.acquire();**

**231. sort\_0\_T6.acquire();**

**232. sort\_1\_T5.acquire();**

**233. sort\_2\_T3.acquire();**

**234. op = new Operations(N, P);**

**235. PS\_1 = op.calcTaskSize(0);**

**236. PS\_2 = op.calcTaskSize(1);**

**237. PS\_3 = op.calcTaskSize(2);**

**238. PS\_4 = op.calcTaskSize(3);**

**239. PS\_5 = op.calcTaskSize(4);**

**240. PS\_6 = op.calcTaskSize(5);**

**241. // static int T2\_sort\_1,T3\_sort\_1,T5\_sort\_1;**

**242. // static int T3\_sort\_2,T5\_sort\_2;**

**243. // static int T2\_sort\_final;**

**244. T2\_sort\_1 = new int[PS\_1+PS\_2];**

**245. T3\_sort\_1 = new int[PS\_3+PS\_4];**

**246. T5\_sort\_1 = new int[PS\_5+PS\_6];**

**247. T3\_sort\_2 = new int[PS\_3+PS\_4+PS\_5+PS\_6];**

**248. T2\_sort\_final = new int[N];**

**249. //start stheads**

**250. Thread t1 = new Thread(new Task1());**

**251. Thread t2 = new Thread(new Task2());**

**252. Thread t3 = new Thread(new Task3());**

**253. Thread t4 = new Thread(new Task4());**

**254. Thread t5 = new Thread(new Task5());**

**255. Thread t6 = new Thread(new Task6());**

**256. t1.start();**

**257. t2.start();**

**258. t3.start();**

**259. t4.start();**

**260. t5.start();**

**261. t6.start();**

**262. t1.join();**

**263. t2.join();**

**264. t3.join();**

**265. t4.join();**

**266. t5.join();**

**267. t6.join();**

**268. }**

**269.**

**270. }**

**1. /\*--------------------------------------------**

**2. -- Operations.java --**

**3. -- A = sort(B\*MO+C\*(MT\*MR) --**

**4. -- Chekanskiy Evgeniy Anatoliyovich, IO-01 --**

**5. -- 5.04.2013 --**

**6. --------------------------------------------\*/**

**7. package crs.cw;**

**8.**

**9. public class Operations {**

**10. private int N;**

**11. private int P;**

**12. public Operations(int N, int P)**

**13. {**

**14. this.N = N;**

**15. this.P = P;**

**16. }**

**17. //розраховує розмір задачі**

**18. public int calcTaskSize(int taskIndex)**

**19. {**

**20. int taskSize = N / P;**

**21. if((taskIndex+1)==P)**

**22. taskSize += N % P;**

**23. return taskSize;**

**24. }**

**25. //множення вектора на матрицю**

**26. public int[] vector\_matrix\_mul\_P(int [] A, int[][] MA, int part)**

**27. {**

**28. int[] result = new int[N];**

**29. int startIndex = part\*(N/P);**

**30. int taskSize = calcTaskSize(part);**

**31. for(int i = startIndex; i < startIndex + taskSize; i++)**

**32. {**

**33. for(int j = 0; j < N; j++)**

**34. {**

**35. result[i] += A[j]\*MA[j][i];**

**36. }**

**37. }**

**38. return result;**

**39. }**

**40. //множення матриць**

**41. public int[][] matrix\_mul\_P(int[][] MA, int[][] MB, int part)**

**42. {**

**43. int[][] result = new int[N][N];**

**44. int startIndex = part\*(N/P);**

**45. int taskSize = calcTaskSize(part);**

**46. for(int i = startIndex; i < startIndex + taskSize; i++)**

**47. {**

**48. for(int j = 0; j < N; j++)**

**49. {**

**50. for(int k = 0; k < N; k++)**

**51. result[j][i] += MA[k][i]\*MB[j][k];**

**52. }**

**53. }**

**54. return result;**

**55. }**

**56. //додавання векторів**

**57. public int[] vector\_add\_P(int[] A,int[] B, int part)**

**58. {**

**59. int[] result = new int[N];**

**60. int startIndex = part\*(N/P);**

**61. int taskSize = calcTaskSize(part);**

**62. for(int i = startIndex; i < startIndex + taskSize; i++)**

**63. {**

**64. result[i] = A[i] + B[i];**

**65. }**

**66. return result;**

**67. }**

**68. public void matrix\_print(int[][] MA)**

**69. {**

**70. for(int i = 0; i < N; i++)**

**71. {**

**72. for(int j = 0; j < N; j++)**

**73. {**

**74. System.out.print(MA[i][j]+" ");**

**75. }**

**76. System.out.println("");**

**77. }**

**78. }**

**79. public void vector\_print(int[] A)**

**80. {**

**81. for(int j = 0; j < N; j++)**

**82. {**

**83. System.out.print(A[j]+" ");**

**84. }**

**85. System.out.println("");**

**86. }**

**87. public int[] vector\_create(int number)**

**88. {**

**89. int[] result = new int[N];**

**90. for(int j = 0; j < N; j++)**

**91. {**

**92. result[j] = number;**

**93. }**

**94. return result;**

**95. }**

**96. public int[][] matrix\_create(int number)**

**97. {**

**98. int[][] result = new int[N][N];**

**99. for(int i = 0; i < N; i++)**

**100. {**

**101. for(int j = 0; j < N; j++)**

**102. {**

**103. result[i][j] = number;**

**104. };**

**105. }**

**106. return result;**

**107. }**

**108. //сортування злиттям**

**109. public void merge\_sort(int[] v1, int[] v2, int[] res, int v1Size, int v2Size)**

**110. {**

**111. int v1Idex = 0, v2Index = 0, resIndex = 0;**

**112. while((v1Idex < v1Size) && (v2Index < v2Size))**

**113. {**

**114. if(v1[v1Idex] <= v2[v2Index])**

**115. {**

**116. res[resIndex] = v1[v1Idex];**

**117. v1Idex++;**

**118. }**

**119. else**

**120. {**

**121. res[resIndex] = v2[v2Index];**

**122. v2Index++;**

**123. }**

**124. resIndex++;**

**125. }**

**126. int[] left;**

**127. int leftIndex;**

**128. if(v1Idex != v1Size)**

**129. {**

**130. left = v1;**

**131. leftIndex = v1Idex;**

**132. }**

**133. else**

**134. {**

**135. left = v2;**

**136. leftIndex = v2Index;**

**137. }**

**138. while(resIndex < v1Size+v2Size)**

**139. {**

**140. res[resIndex] = left[leftIndex];**

**141. leftIndex++;**

**142. resIndex++;**

**143. }**

**144. }**

**145. public int[][] matrix\_copy(int[][] MA)**

**146. {**

**147. int[][] result = new int[MA.length][MA.length];**

**148. for(int i = 0; i < N; i++)**

**149. {**

**150. for(int j = 0; j < N; j++)**

**151. {**

**152. result[i][j] = MA[i][j];**

**153. }**

**154. }**

**155. return result;**

**156. }**

**157. public int[] vector\_copy\_P(int[] A, int part)**

**158. {**

**159. int startIndex = part\*(N/P);**

**160. int taskSize = calcTaskSize(part);**

**161. int[] result = new int[taskSize];**

**162. for(int i = startIndex, j = 0; i < startIndex + taskSize; i++,j++)**

**163. {**

**164. result[j] = A[i];**

**165. }**

**166. return result;**

**167. }**

**168.**

**169. }**

## Додаток Д. Структурна схема ПКС ЛП

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

## Додаток Є. Схеми алгоритмів задач для програми ПРГ2

## Додаток Ж. Лістинг програми ПРГ2

1. ----------------------------------------------

2. -- main.adb --

3. -- A = sort(B\*MO+C\*(MT\*MR) --

4. -- Chekanskiy Evgeniy Anatoliyovich, IO-01 --

5. -- 5.04.2013 --

6. ----------------------------------------------

7. with Ada.Integer\_Text\_IO, Ada.Text\_IO; use Ada.Integer\_Text\_IO, Ada.Text\_IO;

8. with Data; use Data;

9. procedure Main is

10. N,TS,PC : Integer;

11. begin

12. N := 60;

13. TS := N/6;

14. PC := 6;

15. declare

16. task T1 is

17. --передача MO,MT

18. entry PassMOMT(MOx,MTx: in Matrix);

19. --передача MR

20. entry PassMR(MRx: in Matrix);

21. --передача сортування

22. entry PassSort0(S0: In Vector);

23. --передача сортування

24. entry PassSort1Res(S1: In Vector);

25. end T1;

26. task T2 is

27. entry PassBC(Bx,Cx: in Vector);

28. entry PassMOMT(MOx,MTx: in Matrix);

29. entry PassMR(MRx: in Matrix);

30. end T2;

31. task T3 is

32. entry PassBC(Bx,Cx: in Vector);

33. entry PassMR(MRx: in Matrix);

34. end T3;

35. task T4 is

36. entry PassBC(Bx,Cx: in Vector);

37. entry PassMOMT(MOx,MTx: in Matrix);

38. entry PassMR(MRx: in Matrix);

39. entry PassSort0(S0: In Vector);

40. end T4;

41. task T5 is

42. entry PassBC(Bx,Cx: in Vector);

43. entry PassMOMT(MOx,MTx: in Matrix);

44. entry PassSort0Res(S0Res: In Vector);

45. end T5;

46. task T6 is

47. entry PassBC(Bx,Cx: in Vector);

48. entry PassMOMT(MOx,MTx: in Matrix);

49. entry PassMR(MRx: in Matrix);

50. entry PassSort0(S0: In Vector);

51. entry PassSort0Res(S0Res: In Vector);

52. end T6;

53.

54.

55. task body T1 is

56. A,B,C : Vector(0..N);

57. MO,MT : Matrix(0..N\*N);

58. MR: Matrix(0..N\*N);

59. Mtmp1 :Matrix(0..N\*N);

60. Vtmp1, Vtmp2, Vtmp3: Vector(0..N);

61. sortB1,sortB2: Vector(0..(TS-1));

62. sort0Res: Vector(0..(TS\*2-1));

63. sortFinal: Vector(0..(TS\*6-1));

64. begin

65. --Put\_Line("Task 1 started");

66. Mtmp1 := CreateMatrix(N,0);

67. Vtmp1 := CreateVector(N,0);

68. Vtmp2 := CreateVector(N,0);

69. Vtmp3 := CreateVector(N,0);

70. A := CreateVector(N,0);

71. B := CreateVector(N,1);

72. C:= CreateVector(N,1);

73. T2.PassBC(B,C);

74. T6.PassBC(B,C);

75. MO(0):=N;

76. MT(0):=N;

77. MR(0):=N;

78. accept PassMOMT(MOx,MTx: in Matrix) do

79. MVecWRange(MOx,MO,1,TS\*N);

80. MT:=MTx;

81. end PassMOMT;

82. accept PassMR(MRx: in Matrix) do

83. MVecWRange(MRx,MR,1,TS\*N);

84. end PassMR;

85. Matrix\_Mul(MT,MR,Mtmp1,PC,0);

86. Vector\_Matrix\_Mul\_P(C,Mtmp1,Vtmp1,PC,0);

87. Vector\_Matrix\_Mul\_P(B,MO,Vtmp2,PC,0);

88. Vector\_Add(Vtmp1,Vtmp2,Vtmp3);

89. VecRangeSort(Vtmp3,1,TS);

90. --sorting0

91. accept PassSort0(S0: In Vector) do

92. sortB2:=S0;

93. end PassSort0;

94. sortB1 := VecCRange(Vtmp3,1,TS);

95. VecMergeSort(sortB2,sortB1,sort0Res,TS,TS);

96. accept PassSort1Res(S1: In Vector) do

97. VecMergeSort(sort0Res,S1,sortFinal,TS\*2,TS\*4);

98. end PassSort1Res;

99. --Vector\_Print(Vtmp3);

100. Put\_Line("Task 1 ended");

101. end T1;

102.

103.

104. task body T2 is

105. B,C : Vector(0..N);

106. MO,MT : Matrix(0..N\*N);

107. MR: Matrix(0..N\*N);

108. Mtmp1 :Matrix(0..N\*N);

109. Vtmp1, Vtmp2, Vtmp3: Vector(0..N);

110. begin

111. --Put\_Line("Task 2 started");

112. Mtmp1 := CreateMatrix(N,0);

113. Vtmp1 := CreateVector(N,0);

114. Vtmp2 := CreateVector(N,0);

115. Vtmp3 := CreateVector(N,0);

116. B(0) := N;

117. C(0) := N;

118. MO(0):=N;

119. MT(0):=N;

120. MR(0):=N;

121. accept PassBC(Bx,Cx: in Vector) do

122. B:=Bx;

123. C:=Cx;

124. T3.PassBC(B,C);

125. end PassBC;

126. accept PassMOMT(MOx,MTx: in Matrix) do

127. MVecWRange(MOx,MO,1,2\*TS\*N);

128. MT:=MTx;

129. T1.PassMOMT(MVecCRange(MO,1,TS\*N),MT);

130. end PassMOMT;

131. accept PassMR(MRx: in Matrix) do

132. MVecWRange(MRx,MR,1+TS\*N,2\*TS\*N);

133. end PassMR;

134. Matrix\_Mul(MT,MR,Mtmp1,PC,1);

135. Vector\_Matrix\_Mul\_P(C,Mtmp1,Vtmp1,PC,1);

136. Vector\_Matrix\_Mul\_P(B,MO,Vtmp2,PC,1);

137. Vector\_Add(Vtmp1,Vtmp2,Vtmp3);

138. VecRangeSort(Vtmp3,TS+1,2\*TS);

139. T1.PassSort0(VecCRange(Vtmp3,TS+1,2\*TS));

140. --Vector\_Print(Vtmp3);

141. --Put\_Line("Task 2 ended");

142. end T2;

143.

144.

145. task body T3 is

146. B,C : Vector(0..N);

147. MO,MT : Matrix(0..N\*N);

148. MR: Matrix(0..N\*N);

149. Mtmp1 :Matrix(0..N\*N);

150. Vtmp1, Vtmp2, Vtmp3: Vector(0..N);

151. begin

152. --Put\_Line("Task 3 started");

153. Mtmp1 := CreateMatrix(N,0);

154. Vtmp1 := CreateVector(N,0);

155. Vtmp2 := CreateVector(N,0);

156. Vtmp3 := CreateVector(N,0);

157. B(0) := N;

158. C(0) := N;

159. MR(0):=N;

160. MO:=CreateMatrix(N,1);

161. MT:=CreateMatrix(N,1);

162. accept PassBC(Bx,Cx: in Vector) do

163. B:=Bx;

164. C:=Cx;

165. end PassBC;

166. T2.PassMOMT(MVecCRange(MO,1,2\*TS\*N),MT);

167. T4.PassMOMT(MVecCRange(MO,1+3\*TS\*N,6\*TS\*N),MT);

168. accept PassMR(MRx: in Matrix) do

169. MVecWRange(MRx,MR,1+TS\*N,3\*TS\*N);

170. T2.PassMR(MVecCRange(MR,1+TS\*N,2\*TS\*N));

171. end PassMR;

172. --Matrix\_Print(MR);

173. Matrix\_Mul(MT,MR,Mtmp1,PC,2);

174. Vector\_Matrix\_Mul\_P(C,Mtmp1,Vtmp1,PC,2);

175. Vector\_Matrix\_Mul\_P(B,MO,Vtmp2,PC,2);

176. Vector\_Add(Vtmp1,Vtmp2,Vtmp3);

177. VecRangeSort(Vtmp3,2\*TS+1,3\*TS);

178. T4.PassSort0(VecCRange(Vtmp3,2\*TS+1,3\*TS));

179. --Put\_Line("Task 3 ended");

180. end T3;

181.

182.

183. task body T4 is

184. B,C : Vector(0..N);

185. MO,MT : Matrix(0..N\*N);

186. MR: Matrix(0..N\*N);

187. Mtmp1 :Matrix(0..N\*N);

188. Vtmp1, Vtmp2, Vtmp3: Vector(0..N);

189. sortB1,sortB2: Vector(0..(TS-1));

190. sort0Res: Vector(0..(TS\*2-1));

191. begin

192. --Put\_Line("Task 4 started");

193. Mtmp1 := CreateMatrix(N,0);

194. Vtmp1 := CreateVector(N,0);

195. Vtmp2 := CreateVector(N,0);

196. Vtmp3 := CreateVector(N,0);

197. B(0) := N;

198. C(0) := N;

199. MO(0):=N;

200. MT(0):=N;

201. MR(0):=N;

202. accept PassBC(Bx,Cx: in Vector) do

203. B:=Bx;

204. C:=Cx;

205. end PassBC;

206. accept PassMOMT(MOx,MTx: in Matrix) do

207. MVecWRange(MOx,MO,1+3\*TS\*N,6\*TS\*N);

208. MT:=MTx;

209. T5.PassMOMT(MVecCRange(MO,1+4\*TS\*N,6\*TS\*N),MT);

210. --Matrix\_Print(MT);

211. end PassMOMT;

212. accept PassMR(MRx: in Matrix) do

213. MVecWRange(MRx,MR,1+TS\*N,4\*TS\*N);

214. T3.PassMR(MVecCRange(MR,1+TS\*N,3\*TS\*N));

215. end PassMR;

216. --Matrix\_Print(MR);

217. Matrix\_Mul(MT,MR,Mtmp1,PC,3);

218. Vector\_Matrix\_Mul\_P(C,Mtmp1,Vtmp1,PC,3);

219. Vector\_Matrix\_Mul\_P(B,MO,Vtmp2,PC,3);

220. Vector\_Add(Vtmp1,Vtmp2,Vtmp3);

221. VecRangeSort(Vtmp3,3\*TS+1,4\*TS);

222. --sorting0

223. accept PassSort0(S0: In Vector) do

224. sortB2:=S0;

225. end PassSort0;

226. sortB1 := VecCRange(Vtmp3,3\*TS+1,4\*TS);

227. VecMergeSort(sortB2,sortB1,sort0Res,TS,TS);

228. T5.PassSort0Res(sort0Res);

229. --Put\_Line("Task 4 ended");

230. end T4;

231.

232.

233. task body T5 is

234. B,C : Vector(0..N);

235. MO,MT : Matrix(0..N\*N);

236. MR: Matrix(0..N\*N);

237. Mtmp1 :Matrix(0..N\*N);

238. Vtmp1, Vtmp2, Vtmp3: Vector(0..N);

239. begin

240. --Put\_Line("Task 5 started");

241. Mtmp1 := CreateMatrix(N,0);

242. Vtmp1 := CreateVector(N,0);

243. Vtmp2 := CreateVector(N,0);

244. Vtmp3 := CreateVector(N,0);

245. B(0) := N;

246. C(0) := N;

247. MO(0):=N;

248. MT(0):=N;

249. MR:=CreateMatrix(N,1);

250. accept PassBC(Bx,Cx: in Vector) do

251. B:=Bx;

252. C:=Cx;

253. T4.PassBC(B,C);

254. end PassBC;

255. accept PassMOMT(MOx,MTx: in Matrix) do

256. MVecWRange(MOx,MO,1+4\*TS\*N,6\*TS\*N);

257. MT:=MTx;

258. T6.PassMOMT(MVecCRange(MO,1+5\*TS\*N,6\*TS\*N),MT);

259. end PassMOMT;

260. T6.PassMR(MVecCRange(MR,1+5\*TS\*N,6\*TS\*N));

261. T6.PassMR(MVecCRange(MR,1,TS\*N));

262. T4.PassMR(MVecCRange(MR,1+TS\*N,4\*TS\*N));

263. Matrix\_Mul(MT,MR,Mtmp1,PC,4);

264. Vector\_Matrix\_Mul\_P(C,Mtmp1,Vtmp1,PC,4);

265. Vector\_Matrix\_Mul\_P(B,MO,Vtmp2,PC,4);

266. Vector\_Add(Vtmp1,Vtmp2,Vtmp3);

267. VecSet(Vtmp3,4\*TS+3,10);

268. VecRangeSort(Vtmp3,4\*TS+1,5\*TS);

269. T6.PassSort0(VecCRange(Vtmp3,4\*TS+1,5\*TS));

270. accept PassSort0Res(S0Res: In Vector) do

271. T6.PassSort0Res(S0Res);

272. end PassSort0Res;

273. --Put\_Line("Task 5 ended");

274. end T5;

275.

276.

277. task body T6 is

278. B,C : Vector(0..N);

279. MO,MT : Matrix(0..N\*N);

280. MR,MRtmp: Matrix(0..N\*N);

281. Mtmp1 :Matrix(0..N\*N);

282. Vtmp1, Vtmp2, Vtmp3: Vector(0..N);

283. sortB1,sortB2: Vector(0..(TS-1));

284. sort0Res: Vector(0..(TS\*2-1));

285. sort1Res: Vector(0..(TS\*4-1));

286. begin

287. --Put\_Line("Task 6 started");

288. Mtmp1 := CreateMatrix(N,0);

289. Vtmp1 := CreateVector(N,0);

290. Vtmp2 := CreateVector(N,0);

291. Vtmp3 := CreateVector(N,0);

292. B(0) := N;

293. C(0) := N;

294. MO(0):=N;

295. MT(0):=N;

296. MR(0):=N;

297. accept PassBC(Bx,Cx: in Vector) do

298. B:=Bx;

299. C:=Cx;

300. T5.PassBC(B,C);

301. end PassBC;

302. accept PassMOMT(MOx,MTx: in Matrix) do

303. MVecWRange(MOx,MO,1+5\*TS\*N,6\*TS\*N);

304. MT:=MTx;

305. end PassMOMT;

306. accept PassMR(MRx: in Matrix) do

307. MVecWRange(MRx,MR,1+5\*TS\*N,6\*TS\*N);

308. end PassMR;

309. accept PassMR(MRx: in Matrix) do

310. MVecWRange(MRx,MRtmp,1,TS\*N);

311. T1.PassMR(MVecCRange(MRtmp,1,TS\*N));

312. end PassMR;

313. --Matrix\_Print(MR);

314. Matrix\_Mul(MT,MR,Mtmp1,PC,5);

315. Vector\_Matrix\_Mul\_P(C,Mtmp1,Vtmp1,PC,5);

316. Vector\_Matrix\_Mul\_P(B,MO,Vtmp2,PC,5);

317. Vector\_Add(Vtmp1,Vtmp2,Vtmp3);

318. VecRangeSort(Vtmp3,5\*TS+1,6\*TS);

319. --sorting0

320. accept PassSort0(S0: In Vector) do

321. sortB2:=S0;

322. end PassSort0;

323. sortB1 := VecCRange(Vtmp3,5\*TS+1,6\*TS);

324. VecMergeSort(sortB2,sortB1,sort0Res,TS,TS);

325. accept PassSort0Res(S0Res: In Vector) do

326. VecMergeSort(sort0Res,S0Res,sort1Res,TS\*2,TS\*2);

327. T1.PassSort1Res(sort1Res);

328. end PassSort0Res;

329. --Put\_Line("Task 6 ended");

330.

331. end T6;

332. begin

333. null;

334. end;

335. end Main;

1. ----------------------------------------------

2. -- data.ads --

3. -- A = sort(B\*MO+C\*(MT\*MR) --

4. -- Chekanskiy Evgeniy Anatoliyovich, IO-01 --

5. -- 5.04.2013 --

6. ----------------------------------------------

7. package Data is

8. type Vector is array(Integer range <>) of Integer;

9. type Matrix is array(Integer range <>) of Integer;

10. function CreateVector(Size,Number: in Integer) return Vector;

11. function CreateMatrix(Size,Number: in Integer) return Matrix;

12. procedure MatSet(MA: in out Matrix; i,j,Num: in Integer);

13. function MatGet(MA: in Matrix; i,j: in Integer) return Integer;

14. procedure VecSet(A: in out Vector; i,Num: in Integer);

15. function VecGet(A: in Vector; i: in Integer) return Integer;

16. procedure Vector\_Print(A:in Vector);

17. procedure Matrix\_Print(MA:in Matrix);

18. --множення матриць

19. procedure Matrix\_Mul(MA : in Matrix; MB : in Matrix; Result : out Matrix; PC,Part: in Integer);

20. --множення вектора на матрицю

21. procedure Vector\_Matrix\_Mul\_P(A : in Vector; MA : in Matrix; Result : out Vector; PC,Part: in Integer);

22. --додавання векторів

23. procedure Vector\_Add(A,B:in Vector; C : out Vector);

24. --запаковка вектора

25. function VecCRange(A:in Vector; from,to: in Integer) return Vector;

26. --роспаковка вектора

27. procedure VecWRange(A: in Vector;wTo: out Vector; from,to: in Integer);

28. --зпаковка матриці

29. function MVecCRange(A:in Matrix; from,to: in Integer) return Matrix;

30. --роскаковка матриці

31. procedure MVecWRange(A: in Matrix;wTo: out Matrix; from,to: in Integer);

32. --сортування бульбашкою

33. procedure VecRangeSort(A: in out Vector; from,to : in Integer);

34. --сортування злиттям

35. procedure VecMergeSort(v1,v2: in Vector;res: out Vector; v1Size,v2Size: in Integer);

36. end Data;

1. ----------------------------------------------

2. -- data.adb --

3. -- A = sort(B\*MO+C\*(MT\*MR) --

4. -- Chekanskiy Evgeniy Anatoliyovich, IO-01 --

5. -- 5.04.2013 --

6. ----------------------------------------------

7. with Ada.Integer\_Text\_IO, Ada.Text\_IO; use Ada.Integer\_Text\_IO, Ada.Text\_IO;

8. package body Data is

9. function CreateVector(Size,Number: in Integer) return Vector is

10. Result : Vector(0..Size);

11. begin

12. Result(0):=Size;

13. for i in 0..(Size-1) loop

14. VecSet(Result,i,Number);

15. end loop;

16. return Result;

17. end CreateVector;

18. function CreateMatrix(Size,Number: in Integer) return Matrix is

19. Result : Matrix(0..Size\*Size);

20. begin

21. Result(0):=Size;

22. for i in 0..(Size-1) loop

23. for j in 0..(Size-1) loop

24. MatSet(Result,i,j,Number);

25. end loop;

26. end loop;

27. return Result;

28. end CreateMatrix;

29. procedure Vector\_Print(A:in Vector) is

30. begin

31. for Count in 0..A(0)-1 loop

32. Put(VecGet(A,Count));

33. end loop;

34. Put\_Line("");

35. end Vector\_Print;

36. procedure Matrix\_Print(MA: in Matrix) is

37. begin

38. for I in 0..(MA(0)-1) loop

39. for J in 0..(MA(0)-1) loop

40. Put(MatGet(MA,i,j));

41. end loop;

42. Put\_Line("");

43. end loop;

44. end Matrix\_Print;

45. procedure MatSet(MA: in out Matrix; i,j,Num: in Integer) is

46. begin

47. MA(j\*MA(0)+i+1) := Num;

48. end MatSet;

49. function MatGet(MA: in Matrix; i,j: in Integer) return Integer is

50. begin

51. return MA(j\*MA(0)+i+1);

52. end MatGet;

53. procedure VecSet(A: in out Vector; i,Num: in Integer) is

54. begin

55. A(i+1) := Num;

56. end VecSet;

57. function VecGet(A: in Vector; i: in Integer) return Integer is

58. begin

59. return A(i+1);

60. end VecGet;

61.

62. procedure Matrix\_Mul(MA : in Matrix; MB : in Matrix; Result : out Matrix; PC,Part: in Integer) is

63. TaskSize, StartIndex: Integer;

64. begin

65. TaskSize := MA(0) / PC;

66. StartIndex := Part\*TaskSize;

67. if (part+1) = PC then

68. TaskSize := TaskSize + MA(0) mod PC;

69. end if;

70. for i in StartIndex..(StartIndex+TaskSize - 1) loop

71. for j in 0..(MA(0)-1) loop

72. for k in 0..(MA(0) -1) loop

73. MatSet(Result,J,I,MatGet(Result,J,I)+MatGet(MA,K,J)\*MatGet(MB,K,I));

74. end loop;

75. end loop;

76. end loop;

77. end Matrix\_Mul;

78.

79. procedure Vector\_Matrix\_Mul\_P(A : in Vector; MA : in Matrix; Result : out Vector; PC,Part: in Integer) is

80. TaskSize, StartIndex: Integer;

81. begin

82. TaskSize := MA(0) / PC;

83. StartIndex := Part\*TaskSize;

84. if (part+1) = PC then

85. TaskSize := TaskSize + MA(0) mod PC;

86. end if;

87. for i in StartIndex..(StartIndex+TaskSize - 1) loop

88. for j in 0..(MA(0)-1) loop

89. VecSet(Result,i,VecGet(Result,i)+VecGet(A,j)\*MatGet(MA,j,i));

90. end loop;

91. end loop;

92. end Vector\_Matrix\_Mul\_P;

93.

94. procedure Vector\_Add(A,B:in Vector; C : out Vector) is

95. begin

96. for Count in 0..A(0)-1 loop

97. VecSet(C,Count,VecGet(A,Count)+VecGet(B,Count));

98. end loop;

99. end Vector\_Add;

100. function VecCRange(A: in Vector; from,to: in Integer) return Vector is

101. Result : Vector(0..(to-from));

102. counter: Integer;

103. begin

104. counter:=0;

105. for Count in from..to loop

106. Result(counter) := A(Count);

107. counter := counter+1;

108. end loop;

109. return Result;

110. end VecCRange;

111. procedure VecWRange(A: in Vector;wTo: out Vector; from,to: in Integer) is

112. counter: Integer;

113. begin

114. counter:=0;

115. for Count in from..to loop

116. wTo(Count) := A(counter);

117. counter := counter+1;

118. end loop;

119. end VecWRange;

120. function MVecCRange(A: in Matrix; from,to: in Integer) return Matrix is

121. Result : Matrix(0..(to-from));

122. counter: Integer;

123. begin

124. counter:=0;

125. for Count in from..to loop

126. Result(counter) := A(Count);

127. counter := counter+1;

128. end loop;

129. return Result;

130. end MVecCRange;

131. procedure MVecWRange(A: in Matrix;wTo: out Matrix; from,to: in Integer) is

132. counter: Integer;

133. begin

134. counter:=0;

135. for Count in from..to loop

136. wTo(Count) := A(counter);

137. counter := counter+1;

138. end loop;

139. end MVecWRange;

140. procedure VecRangeSort(A: in out Vector; from,to : in Integer) is

141. TMP,Counter: Integer;

142. begin

143. Counter:=0;

144. for J in from..to loop

145. for I in from..(to-Counter-1) loop

146. if A(I) > A(I+1) then

147. TMP := A(I);

148. A(I) := A(I+1);

149. A(I+1) := TMP;

150. end if;

151. end loop;

152. Counter:=Counter+1;

153. end loop;

154. end VecRangeSort;

155. procedure VecMergeSort(v1,v2: in Vector;res: out Vector; v1Size,v2Size: in Integer) is

156. v1Index,v2Index,resIndex: Integer;

157. begin

158. v1Index:=0;

159. v2Index:=0;

160. resIndex:=0;

161. while resIndex < (v1Size+v2Size) loop

162. if v1Index < v1Size and v2Index = v2Size then

163. res(resIndex):= v1(v1Index);

164. v1Index:= v1Index + 1;

165. end if;

166. if v2Index < v2Size and v1Index = v1Size then

167. res(resIndex):= v2(v2Index);

168. v2Index:= v2Index + 1;

169. end if;

170. if v1Index < v1Size and v2Index < v2Size then

171. if v1(v1Index) >= v2(v2Index) then

172. res(resIndex):= v2(v2Index);

173. v2Index:= v2Index + 1;

174. else

175. res(resIndex):= v1(v1Index);

176. v1Index:= v1Index + 1;

177. end if;

178. end if;

179. resIndex:= resIndex+1;

180. end loop;

181. end VecMergeSort;

182. end Data;