НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

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

(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**

з дисципліни «Паралельні та розподілені обчислення»

(назва дисципліни)

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

Студента 3 курсу групи ІО-22

напряму підготовки 050102 «Комп’ютерна інженерія»

Скоріченка О.В.

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

(прізвище та ініціали)

Керівник доцент Корочкін О.В.

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

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_

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

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

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

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

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

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

Київ - 2015 рік

Національний технічний університет України

“Київський політехнічний інститут”

Факультет (інститут) інформатики та обчислювальної техніки

( повна назва )

Кафедра обчислювальної техніки

( повна назва )

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.050102 «Комп’ютерна інженерія»

*(шифр і назва)*

***З А В Д А Н Н Я***

НА КУРСОВУ РОБОТУ СТУДЕНТУ

Скоріченку О.В.

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

*(прізвище, ім’я, по батькові)*

1. Тема роботи «Розробка програмного забезпечення для паралельних

комп’ютерних систем»

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

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

2. Строк подання студентом роботи 18 травня 2014 р.

3. Вхідні дані до роботи

- засоби роботи з процесами в бібліотеках Win32 і POSIX

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

- порівняння засобів роботи з процесами в бібліотеках Win32 і POSIX

- розробка і тестування програми ПРГ1 для ПКС зі СП

- розробка і тестування програми ПРГ2 для ПКС з ЛП

5. Перелік графічного матеріалу

- структурна схема ПКС ОП

- структурна схема ПКС ЛП

- схеми алгоритмів процесів і головної програми для ПРГ1

- схеми алгоритмів процесів і головної програми для ПРГ2.

7. Дата видачі завдання 2.02.2015

***КАЛЕНДАРНИЙ ПЛАН***

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

**Студент \_\_\_\_\_\_\_\_\_** Скоріченко О.В.

( підпис ) (прізвище та ініціали)

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

( підпис ) (прізвище та ініціали)

**ЗМІСТ**

ВСТУП…………………………………………………………………………......5

РОЗДІЛ 1. ПОРІВНЯННЯ ЗАСОБІВ РОБОТИ З ПРОЦЕСАМИ В БІБЛІОТЕКАХ WIN32 І POSIX……………………………………...6

* 1. Огляд засобів бібліотеки POSIX Threads.……………………………6
  2. Огляд засобів бібліотеки Win32.……………………………………11
  3. Висновки до розділу 1……………………………………………….15
     1. Порівняння засобів для створення процесів………………16
     2. Порівняння засобів для вирішення задачі взаємного виключення…………………………………………………..16
     3. Порівняння засобів для вирішення задачі синхронізації…16

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС зі СП………………...18

* 1. Розробка паралельного математичного алгоритму…………………18
  2. Розробка алгоритмів процесів………………………………………..19
  3. Розробка схеми взаємодії процесів………………………………….20
  4. Розробка програми ПРГ1……………………………………………..20
  5. Тестування програми ПРГ1.………………………………………….22
  6. Висновки до розділу 2.………………………………………………..22

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС з ЛП.……..…………..23

* 1. Розробка паралельного математичного алгоритму…………………23
  2. Розробка алгоритмів процесів……………………………………......24
  3. Розробка схеми взаємодії процесів….……………………….………27
  4. Розробка програми ПРГ2……………………………………………..28
  5. Тестування програми ПРГ2………………………………………......30
  6. Висновки до розділу 3………………………………………………...30

ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ДО РОБОТИ………………………30

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……………………………………….31

ДОДАТКИ……………………………………………………..............................32

# ВСТУП

У різних сферах нашого життя для розв’язання багатьох задач необхідна висока продуктивність та висока швидкість передачі інформації, великі об’єми оперативної і постійної пам’яті.

Під терміном паралельні обчислення розуміють сукупність питань, що відносяться до створення ресурсів паралелізму в процесі вирішення задач із метою досягнення більшої ефективності використання обчислювальної техніки, напрямків на збільшення ефективності роботи комп’ютерної системи. Удосконалення комп’ютерів призводить до збільшення їхньої продуктивності, тобто до збільшення можливості виконання більшої кількості операцій. Ефективність паралельної обробки залежить від продуктивності комп’ютерів, від розмірів і структури пам’яті, пропускної здатності каналів зв’язку, використаних мов програмування, компіляторів, операційних систем, чисельних методів та інших математичних досліджень.

У даній курсовій роботі порівнюються засоби роботи з процесами в бібліотеках Win32 і POSIX та розробляються алгоритми обчислення заданого математичного виразу в паралельній комп’ютерній системі зі спільною та локальною пам’яттю. Порівнюються прискорення виконання обчислень та ефективність двох паралельних алгоритмів.

В першому розділі…………..

В другому розділі…………..

В третьому розділі…………..

# РОЗДІЛ 1. ПОРІВНЯННЯ ЗАСОБІВ РОБОТИ З ПРОЦЕСАМИ В БІБЛІОТЕКАХ WIN32 І POSIX

Потоки і процеси - це пов’язані поняття в обчислювальній техніці. Обидва представляють із себе послідовність інструкцій, які повинні виконуватися в певному порядку, але інструкції в окремих потоках або процесах можуть виконуватися паралельно.

Процеси існують в операційній системі і відповідають тому, що користувачі бачать як програми або додатки. З іншого боку, потік існує всередині процесу. Потік (thread) визначає послідовність виконання коду у процесі [1]. З цієї причини потоки іноді називаються «легкі процеси».

Кожен процес складається з одного або більше потоків. Існування кількох процесів дозволяє комп'ютеру «одночасно» виконувати декілька завдань. Існування декількох потоків дозволяє процесу розділяти роботу для паралельного виконання. На багатопроцесорному комп’ютері процеси або потоки можуть працювати на різних процесорах, що дозволяє виконувати реально паралельну роботу.

Проте абсолютно паралельна обробка не завжди можлива, адже потоки іноді повинні синхронізуватися. Один потік може очікувати результату іншого потоку, або одному потоку може знадобитися монопольний доступ до ресурсу, який використовується іншим потоком. Проблеми синхронізації є поширеною причиною помилок у багатопоточних додатках.

Прикладом бібліотечних засобів роботи з процесами є засоби бібліотек WinAPI (Win32), PVM, POSIX Threads (Pthreads), MPI, OpenMP.

Відповідно до технічого завдання розглянуті та порівняні засоби роботи з потоками бібліотек Win32 та POSIX Threads.

**1.1 Огляд засобів бібліотеки POSIX Threads**

POSIX Threads - стандарт POSIX реалізації потоків (ниток) виконання. Стандарт POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995) визначає API для управління потоками, їх синхронізації і планування.

Реалізації даного API існують для великого числа UNIX-подібних ОС (Linux, Solaris, FreeBSD, OpenBSD, NetBSD, OS X), а також для Microsoft Windows та інших ОС. Бібліотеки, що реалізують цей стандарт (і функції цього стандарту), зазвичай називаються Pthreads (функції мають приставку «pthread\_»). Pthreads визначає набір типів даних, функцій і констант в форматі мови програмування C. Вони описані в файлі заголовку pthread.h і реалізовані у вигляді бібліотеки.

POSIX API для семафорів працює з потоками POSIX, але не є частиною стандарту роботи з потоками, визначається в додатковому стандарті POSIX.1b, Real-time extensions (IEEE Std 1003.1b-1993). Всі процедури для роботи з семафорами, відповідно, мають префікс "sem\_" замість "pthread\_" [3].

int pthread\_create(pthread\_t \*newthread, const pthread\_attr\_t \*attr, void \*(\*start\_routine)(void\*), void \*arg);

Перший параметр цієї функції являє собою вказівник на змінну типу pthread\_t, яка служить ідентифікатором створюваного потоку. Другий параметр, покажчик на змінну типу pthread\_attr\_t, використовується для передачі атрибутів потоку. Третім параметром функції pthread\_create() є адреса функції потоку. Четвертий параметр функції pthread\_create() має тип void \*. Цей параметр може використовуватися для передачі значення, що повертається функцією потоку.

#include <pthread.h>

#include <unistd.h>

#define nil NULL

void v1(void) {…

}

void v2(void) {…

}

int main(int argc, const char \*argv[]) {

pthread\_t t1, t2;

pthread\_create(&t1, nil, (void \*(\*)(void \*)) v1, nil);

pthread\_create(&t2, nil, (void \*(\*)(void \*)) v2, nil);

pthread\_join(t1, nil);

pthread\_join(t2, nil);

return 0;

}

Відразу після того, як був викликаний pthread\_create, потік відокремлюється і починає виконувати свою функцію. Тобто все, що після pthread\_create відбувається у функції - відбувається паралельно main потоку (у тому якому виконується головна main функція). Функція pthread\_join() призупиняє виконання потоку, що її викликав до тих пір, поки потік, чий ідентифікатор переданий функції як аргумент, не завершить свою роботу.

#include <рthread.h>

int pthread\_join(pthread\_t th, void\*\* thread\_return);

Коли потік завершується, він викликає функцію pthread\_exit, що завершує викликаний потік, повертаючи покажчик на об'єкт [4]:

#include <рthread.h>

void pthread\_exit(void \*retval);

Потік в pthreads може бути скасований (cancel), тобто просто знятий з виконання допомогою системного виклику pthread\_cancel в момент очікування звільнення якого-небудь мютекса або умовної змінної. Іноді потрібно, щоб один потік попросив інший завершитися достроковим способом, дуже схожим на відправку йому сигналу:

#include <pthread.h>

int pthread\_cancel(pthread\_t thread);

Для роботи з мютексами і семафорами існують окремі виклики - pthread\_mutex\_lock/pthread\_mutex\_unlock і т.д. Потік pthread не може перебувати в стані сну. Також в pthread немає подій, але є набагато більш потужний механізм – механізм умовних змінних (conditional variables).

Умовна змінна - об'єкт в системі, який може використовуватися для оповіщення потоків про настання якоїсь умови. Умовна змінна повинна бути ініціалізована функцією:

int pthread\_cond\_init(pthread\_cond\_t \*cond, const pthread\_condattr\_t \*attr);

Ресурси, зайняті нею, можуть бути звільнені функцією:

int pthread\_cond\_destroy(pthread\_cond\_t \*cond);

Функції роботи з умовними змінними:

int pthread\_cond\_wait(pthread\_cond\_t \*cond, pthread\_mutex\_t \*mutex);

* призупиняє потік до настання події cond. Мютекс звільняється на час очікування і знову займається при виході з функції;

int pthread\_cond\_signal(pthread\_cond\_t \*cond);

* сигналізує про настання події cond. При цьому тільки один з потоків, які очікують подія (якщо є такий), вийде з очікування;

int pthread\_cond\_broadcast(pthread\_cond\_t \*cond);

* сигналізує про настання події cond. При цьому всі потоки, які очікують подія (якщо є такі), виходять з очікування [5].

Для семафорів є два набори інтерфейсних функцій: один узятий з POSIX Realtime Extensions (доповнення POSIX для режиму реального часу) і застосовується для потоків, а інший, відомий як семафори System V, що зазвичай застосовується для синхронізації процесів. Обидва набору не гарантують взаємозамінності і хоча дуже схожі, використовують виклики різних функцій.

Голландський учений, фахівець з комп'ютерних наук Дейкстра першим сформулював ідею семафорів. Семафор - це змінна особливого типу, яка може змінюватися з позитивним чи негативним приростом, але звернення до змінної у відповідний момент завжди атомарне навіть в багатопоточних програмах [4].

Для роботи з потоками застосовують базові функцій семафорів. Семафор створюється за допомогою функції sem\_init, яка оголошується:

#include <semaphore.h>

int sem\_init(sem\_t \*sem, int pshared, unsigned int value);

Ця функція ініціалізує об'єкт-семафор, на який вказує параметр sem, що задає варіант його спільного використання і присвоює йому початкове цілочисельне значення. Параметр pshared управляє типом семафора. Якщо pshared дорівнює 0 - семафор локальний по відношенню до поточного процесу. В іншому випадку семафор може бути спільно використаний різними процесами.

Наступна пара функцій управляє значенням семафора і оголошується наступним чином:

#include <semaphore.h>

int sem\_wait(sem\_t \*sem);

int sem\_post(sem\_t \*sem);

Функція sem\_post атомарно збільшує значення семафора на 1.

Функція sem\_wait атомарно зменшує значення семафора на одиницю, але завжди чекає до тих пір, поки спочатку лічильник семафора не отримає ненульове значення.

Функція семафорів sem\_destroy очищає семафор, коли робота з ним закінчена і оголошується:

#include <semaphore.h>

int sem\_destroy(gem\_t \*sem);

Інший спосіб синхронізації доступу в багатопоточних програмах - застосування мютексів (скорочення від mutual exclusions - взаємні виключення), які дозволяють програмістам «замикати» об'єкт так, що тільки один потік може звернутися до нього. Мютекс повинен бути обов’язково ініціалізований перед використанням:

#include <рthread.h>

int pthread\_mutex\_init(pthread\_mutex\_t \*mutex,

const pthread\_mutexattr\_t \*mutexattr);

Параметр attr задає атрибути мютекса (можливо передати NULL для прийняття атрибутів за замовчуванням). Ресурси, що зайняті мютексом, можуть бути звільнені функцією:

int pthread\_mutex\_destroy(pthread\_mutex\_t \*mutex);

Для захоплення мютекса потік використовує функцію:

int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);

а для звільнення [6]:

int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

У паралельних обчисленнях бар’єром називають метод синхронізації. Бар’єр для групи потоків або процесів у вихідному коді значить, що кожний потік/процес мають зупинитися у цій точці і не можуть продовжуватися, доки всі потоки/процеси не досягнуть цієї точки. У бібліотеці Pthreads існує можливість роботи з бар’єрами [4].

Ініціалізація бар’єру:

#include <pthread.h>

int pthread\_barrier\_init(pthread\_barrier\_t \*restrict barrier, pthread\_barrierattr\_t \*restrict attr, unsigned count);

Знищення бар’єру та звільнення ресурсів:

#include <pthread.h>

int pthread\_barrier\_destroy(pthread\_barrier\_t \*barrier);

Синхронізація потоків - викликаючий потік блокується, поки потрібна кількість потоків не прийде до бар’єру:

#include <pthread.h>

int pthread\_barrier\_wait(pthread\_barrier\_t \*barrier);

**1.2 Огляд засобів бібліотеки Win32**

HANDLE CreateThread(

LPSECURITY\_ATTRIBUTES lpThreadAttributes, //атрибут безпеки

DWORD dwStackSize, //розмір стека

LPTHREAD\_START\_ROUTINE lpStartAddress, //потокова функція

LPVOID lpParameter, //аргумент функції потоку

DWORD dwCreationFlags, //прапорець виконання

LPDWORD lpThreadId); //ідентифікатор потоку

Потоки в Win32 створюються за допомогою функції CreateThread, куди передається покажчик на функцію потоку, яка буде виконуватися у створеному потоці. Потік вважається завершеним, коли виконається функція потоку. Функція ExitThread буде викликана неявно, коли завершиться функція потоку, або ж її можна викликати самостійно. Головне її завдання - звільнити стек потоку і структури ядра, які обслуговують даний потік [1].

Потік в Win32 може перебувати в стані сну (suspend). Можна «приспати потік» за допомогою виклику функції SuspendThread, і «розбудити» його за допомогою виклику ResumeThread, також потік можна перевести в стан сну при створенні, встановивши значення параметра СreateSuspended функції CreateThread. Pthreads не підтримують подібну функціональність.

Засоби сінхроніхаціі в Win32 є двох типів: реалізовані на рівні користувача, і на рівні ядра. Перші - це критичні секції (critical section), до другого набору відносять мютекси (mutex), події (event) і семафори (semaphore).

Критичні секції - легкий механізм синхронізації, який працює на рівні користувача процесу і не використовує важких системних викликів. Він заснований на механізмі взаємних блокувань. Потік, який бажає убезпечити певні дані викликає функцію EnterCliticalSection/TryEnterCriticalSection. Якщо критична секція вільна - потік займає її, якщо ж ні - потік блокується (тобто не виконується і не забирає процесорний час) до тих пір, поки секція не буде звільнена іншим потоком за допомогою виклику функції LeaveCriticalSection.

У мютексів, подій та семафорів є спільні риси:

Вони використовують примітиви ядра при виконанні, тобто системні виклики, що позначається на продуктивності;

Можуть бути іменованими і неіменованими, тобто кожному такому об'єкту синхронізації можна присвоїти ім'я;

Працюють на рівні системи, а не на рівні процесу, тобто можуть служити механізмом взаємодії між процесами;

Використовують для очікування і захоплення примітиву єдину функцію: WaitForSingleObject / WaitForMultipleObjects.

У бібліотеці Win32 після закінчення роботи потоку необхідно закрити дескриптори функцією CloseHandle(). Блокування (призупинення) процесу виконується за допомогою функцій SuspendThread() та ResumeThread() [2].

Для вирішення завдання взаємного виключення можна використати механізми семафорів, мютексів, критичних секцій.

Семафор являє собою лічильник від нуля до визначеного значення, яке визначає кількість потоків, що мають доступ до спільного ресурсу.

Створення семафору забезпечується викликом функції:

s = CreateSemaphore(NULL, 0, 1, NULL);

Під час ініціалізації встановлюється максимальне значення семафора (1 – для бінарних семафорів). Функцією очікування є WaitForSingleObject(). Вона перевіряє значення семафора: якщо значення дорівнює 0 – ппроцес, що викликає функцію – блокується, інакше – відбувається інкремент лічильника. Функцією встановлення семафору є ReleaseSemaphore() [2].

WaitForSingleObject(s, INFINITE); (значення INFINITE – показує, що по процес буде блокованим поки значення семафора не зміниться).

ReleaseSemaphore(s, 1, NULL); (збільшує лічильник семафора на визначене значення).

Функція WaitForMultipleObject() – множинна функція очікування. Вона дозволяє потоку, що викликає визначити масив, який містить декілька ідентифікаторів об’єктів синхронізації.

DWORD Tid1, Tid2;

HANDLE threads[] = {

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)T1, NULL, NULL, &Tid1),

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)T2, NULL, NULL, &Tid2)};

WaitForMultipleObjects(2, threads, TRUE, INFINITE);

CloseHandle(threads[0]);

CloseHandle(threads[1]);

Мютекс має два стани: вільний та захоплений процесом. Він може належати тільки одному процесу, який захопив мютекс та має право на вхід в критичну ділянку. Створення мютексу виконується зі зміною типу HANDLE за допомогою операцій CreateMutex(). Критична ділянка обрамлюється двома функціями, першою з яких є функція очікування WaitForSingleObject(), а другою ReleaseMutex() [2]. Приклад мютекса та критичної ділянки:

WaitForSingleObject(mutex, INFINITE);

a1 = a;

ReleaseMutex(mutex);

Реалізація механізму критичних секцій у бібліотеці Win32 подібна до реалізації мютексів та семафорів. Функції критичної секції після її ініціалізації контролюють входження та виходження процесу з критичної ділянки. Створення критичної секції виконується зі змінною типу CRITICAL\_SECTION [2]:

InitializeCriticalSection(&cs);

DeleteCriticalSection() – операція знищення критичної секції.

EnterCriticalSection(&cs); //вхід в критичну ділянку

a1 = a;

LeaveCriticalSection(&cs); //вихід з критичної ділянки

Задачу синхронізації процесів можна вирішити за допомогою семафорів та подій.

При використанні механізму семафорів функція WaitForSingleObject() забезпечує очікування події та блокування потоку, що її викликає у разі ненастання події. Функція ReleaseSemaphore() використовується для встановлення сигналу про подію, що відбулася. При вирішенні проблем синхронізації доцільно використовувати множинні семафори, що можуть набувати різні значення [2]. При ініціалізації множинного семафора вказується його максимальне значення, наприклад:

sm = CreateSemaphore(NULL, 0, 2, NULL);

Механізм подій побудований на перемиканні станів: сигнального та несигнального. Після певної події процес встановлює сигнальний стан за допомогою функції SetEvent() - сповіщує інший процес про подію. Очікування на подію відбувається за допомогою функції WaitForSingleObject(). Для сповіщення одразу кількох процесів існує режим ручного скидання, що встановлюється під час ініціалізації події. Він дозволяє не змінювати стан події після того, як один з процесів її перевірив, тобто сигнал проходить далі до наступних процесів [2].

Evv = CreateEvent(NULL, TRUE, 0, NULL);

**1.3 Висновки до розділу 1**

У Pthreads кожен об'єкт має свій власний тип даних (pthread\_t, pthread\_mutex\_t, pthread\_cond\_t і т.д.), а у потоках Win32 є один тип: HANDLE. Pthreads використовує різні функції для роботи з кожним типом об'єкта, тож програміст повинен знати число, порядок і тип параметрів для всіх різних функцій. З іншого боку єдиний тип HANDLE спрощує роботу з об’єктами у паралельні програмі та дозволяє програмісту пам’ятати фіксовану кількість параметрів, що спрощує його дії. Можливо, найбільша перевага одного типу даних об'єкта у тому, що необхідна лише одна функція, щоб блокувати потік під час очікування об'єкта: WaitForSingleObject. Таким чином, лише один набір параметрів має бути відомий незалежно від того, чи чекає код в потоці мютекса, семафора або події. З іншого боку, використання одного типу даних для різних об’єктів може призвести до проблем та плутанини при використанні в WaitFor\* викликів, адже програміст повинен задати ім'я HANDLE так, щоб це очевидно сприймалось і було видно, чи код очікує на припинення потоку, сигнал про подію, або звільнення мютекса.

**1.3.1 Порівняння засобів для створення процесів**

У бібліотеці Pthreads та Win32 потоки створюються за допомогою виклику функції create. У якості параметрів передаються атрибути, потокова функція, ідентифікатор потоку. Проте бібліотека Win32 дозволяє створити потік «у стані сну» (suspend) – відкласти його виконання. Його можна призупинити за допомогою виклику функції SuspendThread, і «розбудити» його за допомогою виклику ResumeThread. Обидві бібліотеки мають функцію join, що забезпечує блокування потоку доки потік, чий ідентифікатор переданий функції як аргумент, не завершить свою роботу. Існують функції завершення потоку у обох бібліотеках.

**1.3.2 Порівняння засобів для вирішення задачі взаємного виключення**

Обидві бібліотеки реалізують механізм мютексів. Однак бібліотека Win32 має додатково механізм критичних секцій для вирішення задачі взаємного виключення.

**1.3.3 Порівняння засобів для вирішення задачі синхронізації**

У Pthreads семафори виділені в окрему бібліотеку та реалізовані подібно до бібліотеки Win32. Використовуються атомарні операції для збільшення та зменшення значення семафора. У бібліотеці Win32 можливо використовувати множинні семафори.

Множинна функція очікування WaitForMultipleObject() у Win32 легка у використанні та долає проблему окремого виклику функцій очікування, а забезпечує очікування відразу кількох об’єктів синхронізації.

Бібліотека Pthreads не має подій, проте має гнучкий механізм умовних змінних. Події у Win32 дозволяють сповіщати одразу декілька процесів, використовуючи опцію ручного скидання.

У бібліотеці Win32 у механізмах мютексів та подій відбувається збереження станів: вільний/захоплений, сигнальний/несигнальний. Їх можна скинути вручну. У бібліотеці Pthreads існує можливість пропустити сигнал про подію, який не зберігається.

Переваги бібліотеки Pthreads полягають у наявності механізму бар’єра, а також у можливості налаштування умовних змінних за часом очікування.

**РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС зі CП**

У даному розділі розглянуто розробку та дослідження програми ПРГ1 для ПКС зі СП.

Математична задача: MA = max(E)\*MO\*MK + alfa\*MM.

Мова програмування: С++.

Засоби взаємодії: бібліотека OpenMP (бар’єри, критичні секції).

Структуру ПКС зі СП представлено на рис. 2.1.



Рис. 2.1 Структура ПКС зі СП для р-процесорів

**2.1 Розробка паралельного математичного алгоритму**

Позначення, що використовуються:

n - розмірність векторів та матриць;

p - кількість процесорів;

H = n/p;

1. ei = max(EH) (i = 1..p)
2. e = max(e, ei)

СР: e

1. MAH = e\*MOH\*MK + alfa\*MMH

СР: e, MK, alfa

**2.2 Розробка алгоритмів процесів**

|  |  |
| --- | --- |
| **T1** | **ТС КД** |
| 1. Ввести дані: E, MK, alfa 2. Сигнал T2…Tp про введення даних 3. Очікувати введення даних в Tp 4. e1:=max(EH) 5. e:=max(e, e1) 6. Сигнал T2…Tp про завершення обчислення 7. Очікувати завершення обчислення в T2…Tp 8. Копіювати: e1:=e, MK1:=MK, alfa1:=alfa 9. Обчислити: MAH:=e1\*MOH\*MK1 + alfa1\*MMH 10. Очікувати завершення обчислення в T2…Tp 11. Вивести дані: MА | S2…p,1  Wp,1  КД  S2…p,2  W2…p-1,1 Wp,2  КД  W2…p-1,2 Wp,3 |

|  |  |
| --- | --- |
| **Ti (1<i<p)** | **ТС КД** |
| 1. Очікувати введення даних в Т1, Tp 2. ei:=max(EH) 3. e:=max(e, ei) 4. Сигнал T1…Ti-1, Ti+1…Tpпро завершення обчислення 5. Очікувати завершення обчислення в T1…Ti-1, Ti+1…Tp 6. Копіювати: ei:=e, MKi:=MK, alfai:=alfa 7. Обчислити: MAH:=ei\*MOH\*MKi + alfai\*MMH 8. Сигнал T1про завершення обчислення | W1,1 Wp,1  КД  S1…i-1,1 Si+1…p,1  W2…i-1,1 Wi+1…p-1,1 W1,2 Wp,2  КД  S1,2 |

|  |  |
| --- | --- |
| **Tp** | **ТС КД** |
| 1. Ввести дані: MO, MM 2. Сигнал T1…Tp-1про введення даних 3. Очікувати введення даних в T1 4. ep:=max(EH) 5. e:=max(e, ep) 6. Сигнал T1…Tp-1про завершення обчислення 7. Очікувати завершення обчислення в T1…Tp-1 8. Копіювати: ep:=e, MKp:=MK, alfap:=alfa 9. Обчислити: MAH:=ep\*MOH\*MKp + alfap\*MMH 10. Сигнал T1про завершення обчислення | S1…p-1,1  W1,1  КД  S1…p-1,2  W1,2 W2…p-1,1  КД  S1,3 |

**2.3 Розробка схеми взаємодії процесів**

Введення даних відбувається у першому (T1) та в останньому (Tp) процесах. Синхронізація по введенню виконується за допомогою механізму бар’єрів, що є у бібліотеці OpenMP (#pragma omp barrier).

Задача взаємного виключення та доступу до спільних ресурсів вирішується за допомогою критичних секцій (#pragma omp critical):

* cs1 - критична секція для пошуку максимума (СР: e);
* cs2 - критична секція для копіювання спільних ресурсів (СР: e, MK, alfa).

Також за допомогою бар’єрів виконується синхронізація між процесами після пошуку максимального елемента у векторі, а також по виведенню.

Матрицю виводить процес T1.

Схема взаємодії процесів зображена на рис. 2.2.

**2.4 Розробка програми ПРГ1**

Бібліотека OpenMP дозволяє легко створювати паралельні програми за допомогою #pragma omp parallel. При виконанні створюються копії тіла прагми і вони виконуються паралельно. Кожен процес (потік) отримує свій ідентифікатор. Це дозволяє використати його для вибору індивідуальних дій (введення/виведення даних) для кожного потоку, що спрощує написання програми та зводить структуру коду до загального вигляду з операторами вибору if().



Рис. 2.2 Схема взаємодії процесів

Для спрощення розпаралелювання обчислення заданого математичного виразу використовується важлива особливість бібліотеки OpenMP – паралельні цикли (#pragma omp for). Цей механізм самостійно розділяє вектори та матриці на частини для паралельної обробки усіма процесорами.

Для створення векторів, матриць, а також виведення матриці використовуються процедури: generateVector(), generateMatrix(), outMatrix(), відповідно.

Отримання значень часу роботи програми реалізовано за допомогою бібліотеки <time.h>.

Лістинг ПРГ1 знаходиться у Додатку А.

Блок-схема алгоритму головної процедури ПРГ1 та алгоритм головної програми ПРГ1 знаходяться у Додатках В та Г.

**2.5 Тестування програми ПРГ1**

(10.05.2015)

**2.6 Висновки до розділу 2**

Виконано розробку програми ПРГ1 для ПКС зі СП. При розробці використовувалась мова програмування C++ з бібліотекою OpenMP. Для синхронізації процесів використовувався механізм бар’єрів.

1

2

3

4

5

**РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС з ЛП**

У даному розділі розглянуто розробку та дослідження програми ПРГ2 для ПКС з ЛП.

Математична задача: MA = max(E)\*MO\*MK + alfa\*MM.

Мова програмування: С++.

Засоби взаємодії: бібліотека MPI.

Структуру ПКС з ЛП представлено на рис. 3.1.



Рис. 3.1 Структура ПКС з ЛП для р-процесорів

**3.1 Розробка паралельного математичного алгоритму**

Позначення, що використовуються:

n - розмірність векторів та матриць;

p - кількість процесорів;

H = n/p;

Q = p/2;

1. ei = max(EH) (i = 1..p)
2. e = max(e, ei)
3. MAH = e\*MOH\*MK + alfa\*MMH
   1. **Розробка алгоритмів процесів**

|  |
| --- |
| **Т1** |
| 1. Введення MK, E, alfa 2. Передати MK, alfa, EN-2H в Т2 3. Прийняти MO2H, MM2H від T2 4. Передати MK, alfa, EH, MOH, MMH в Т1.1 5. Обчислення e1:=max(EH) 6. Прийняти e1.1 від Т1.1 7. Обчислення e1:=max(e1, eQ.1) 8. Передати e1 в Т2 9. Прийняти e1 від Т2 10. Передати e1 в Т1.1 11. Обчислення MAH:=e\*MOH\*MK + alfa\*MMH 12. Прийняти МАН від Т1.1 13. Прийняти МАN-2H від Т2 14. Виведення МА. |
| **ТQ, Q = (P/4) + 1, Q ≠ P/2** |
| 1. Прийняти MO2HQ, MM2HQ від ТQ+1 2. Передати MO2H(Q-1), MM2H(Q-1) в ТQ-1 3. Прийняти MK, alfa, E(N-2Q)H від ТQ-1 4. Передати MK, alfa, E(N-2Q-2)H ­в ТQ+1 5. Передати MK, MMH, EH, alfa, MOH в ТQ.1 6. Обчислення eQ:=max(EH) 7. Прийняти eQ.1 від ТQ.1 8. Прийняти eQ+1 від ТQ+1 9. Прийняти eQ-1 від ТQ-1 10. Обчислення eQ:=max(eQ, eQ.1, e Q+1, e Q-1) 11. Передати e1 в ТQ+1 12. Передати e1 в ТQ-1 13. Передати e1 в ТQ.1 14. Обчислення MAH:=e\*MOH\*MK + alfa\*MMH 15. Прийняти МАН від ТQ.1 16. Прийняти МА2H(Q-1) від ТQ+1 17. Передати МА2HQ в ТQ-1 |
| **ТQ, Q = P/2, Q ≠ 2** |
| 1. Введення MO, MM 2. Передати MON-2H, MMN-2H в ТQ-1 3. Прийняти MK, alfa, E2H від ТQ-1 4. Передати MK, MMH, EH, alfa, MOH в ТQ.1 5. Обчислення eQ:=max(EH) 6. Прийняти eQ.1 від ТQ.1 7. Обчислення eQ:=max(eQ, eQ.1) 8. Передати eQ в ТQ-1 9. Прийняти eQ від ТQ-1 10. Передати eQ в ТQ.1 11. Обчислення MAH:=e\*MOH\*MK + alfa\*MMH 12. Прийняти МАН від ТQ.1 13. Передати МА2H в ТQ-1 |
| **ТQ, Q = P/2, Q = 2** |
| 1. Введення MO, MM 2. Передати MON-2H, MMN-2H в ТQ-1 3. Прийняти MK, alfa, E2H від ТQ-1 4. Передати MK, MMH, EH, alfa, MOH в ТQ.1 5. Обчислення eQ:=max(EH) 6. Прийняти eQ.1 від ТQ.1 7. Прийняти eQ-1 від ТQ-1 8. Обчислення eQ:=max(eQ, eQ.1, eQ-1) 9. Передати eQ в ТQ-1 10. Передати eQ в ТQ.1 11. Обчислення MAH:=e\*MOH\*MK + alfa\*MMH 12. Прийняти МАН від ТQ.1 13. Передати МА2H в ТQ-1 |
| **ТQ, Q < (P/4) + 1, Q ≠ 1** |
| 1. Прийняти MK, alfa, E(N-2Q)H від ТQ-1 2. Передати MK, alfa, E(N-2Q-2)H ­в ТQ+1 3. Прийняти MO2HQ, MM2HQ від ТQ+1 4. Передати MO2H(Q-1), MM2H(Q-1) в ТQ-1 5. Передати MK, MMH, EH, alfa, MOH в ТQ.1 6. Обчислення eQ:=max(EH) 7. Прийняти eQ.1 від ТQ.1 8. Прийняти e Q-1 від ТQ-1 9. Обчислення eQ:=max(eQ, eQ.1, e Q-1) 10. Передати eQ в ТQ+1 11. Прийняти eQ від ТQ+1 12. Передати eQ в ТQ-1 13. Передати eQ в ТQ.1 14. Обчислення MAH:=e\*MOH\*MK + alfa\*MMH 15. Прийняти МАН від ТQ.1 16. Прийняти МА2H(Q-1) від ТQ+1 17. Передати МА2HQ в ТQ-1 |
| **ТQ, Q > (P/4) + 1, Q ≠ P/2** |
| 1. Прийняти MO2HQ, MM2HQ від ТQ+1 2. Передати MO2H(Q-1), MM2H(Q-1) в ТQ-1 3. Прийняти MK, alfa, E(N-2Q)H від ТQ-1 4. Передати MK, alfa, E(N-2Q-2)H ­в ТQ+1 5. Передати MK, MMH, EH, alfa, MOH в ТQ.1 6. Обчислення eQ:=max(EH) 7. Прийняти eQ.1 від ТQ.1 8. Прийняти eQ-1 від ТQ+1 9. Обчислення eQ=max(eQ, eQ.1, e Q+1) 10. Передати eQ в ТQ-1 11. Прийняти eQ від ТQ-1 12. Передати eQ в ТQ+1 13. Передати eQ в ТQ.1 14. Обчислення MAH:=e\*MOH\*MK + alfa\*MMH 15. Прийняти МАН від ТQ.1 16. Прийняти МА2H(Q-1) від ТQ+1 17. Передати МА2HQ в ТQ-1 |
| **ТQ.1, Q = 1..P/2** |
| 1. Прийняти MK, MMH, EH, alfa, MOH від ТQ 2. Обчислення eQ:=max(EH) 3. Передати eQ в ТQ 4. Прийняти eQ від ТQ 5. Обчислення MAH:=e\*MOH\*MK + alfa\*MMH 6. Передати МАH в ТQ |

**3.3 Розробка схеми взаємодії процесів**

Усі задачі логічно розбиваються на декілька типів. Таке логічне розбиття на типи гарантує, що пересилка даних та збір і розповсюдження проміжних результатів буде виконуватися за кількість кроків, що дорівнює діаметру системи. Тобто, найбільш оптимально:

* Перша задача - T1, в якій вводяться дані (E, MK, alfa).
* Остання задача - TQ, в якій вводяться дані (MO, MM).
* «Проміжна» (середня) задача ТQ, Q = (P/4) + 1, у якій збирається результат
* «Проміжні» задачі ліворуч від середньої - ТQ, Q < (P/4) + 1, Q ≠ 1.
* «Проміжні» задачі праворуч від середньої - ТQ, Q > (P/4) + 1, Q ≠ P/2.
* «Нижні» задачі ТQ.1, Q = 1..P/2.

Передача повідомлень між процесами та синхронізація відбувається за допомогою викликів таких функцій:

* MPI\_Recv – функція блокує процес поки не будуть отриманні всі дані;
* MPI\_Send – функція блокує процес поки не будуть передані всі дані;
* MPI\_Isend – функція викликає передачу даних без блокування процесу; Схема взаємодії процесів зображена на рис. 3.2.

**3.4 Розробка програми ПРГ2**

Програма ПРГ2 розроблена на мові програмування C++ із використанням засобів бібліотеки MPI.

Програма складається з файлів Matrix.h, Matrix.cpp, CourseWork(mpi).cpp. У файлі Matrix.h розміщується об’явлення класу Matrix та його конструкторів і функцій. У файлі Matrix.cpp розміщується визначення конструкторів і функцій класу Matrix:

input() – заповнення елементів матриці значенням константи FILLER;

copy(Matrix& copyMatrix) – копіює матрицю;

mult(Matrix& res, Matrix& multMatr) – множення матриць;

getMax() – пошук максимуму;

getMax(int from, int to) – пошук максимуму у певній ділянці вектора;

delExcessive(int from, int to) – видалення усіх елементів, окрім діапазону, вказаному в аргументах. Це необхідно для зменшення використання пам’яті при пересилках матриць. Після закінчення передачі матриці, виконується видалення зайвих елементів.

merge(Matrix& MAh) – об’єднання двох матриць.

У класі CourseWork(mpi).cpp визначений метод main(), в якому запускаються процеси, а також константи n, p, H, допоміжні функції та процеси. Лістинг розробленої програми наведено у додатку Б. Алгоритм роботи задач наведено у додатку Д.

?



Рис. 3.2 Схема взаємодії процесів ПОД РИСУНОК!!!!!

**3.5 Тестування програми ПРГ2**

(10.05.2015)

**3.6 Висновки до розділу 3**

Виконано розробку програми ПРГ2 для ПКС з ЛП. При розробці використовувалась мова програмування C++ з бібліотекою MPI.

1

2

3

4

5

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows / Пер. с англ. - 4е изд. - Спб.: Питер; М.: Издательство «Русская Редакция»; 2008. - 720 стр.
2. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення: Навч. посіб. – 2-ге вид. виправл. і доп. К.: Корнійчук, 2014. – 284 с.
3. Why Windows Threads Are Better Than POSIX Threads [Електронний ресурс]. – https://software.intel.com/en-us/blogs/2006/10/19/why-windows-threads-are-better-than-posix-threads
4. Потоки POSIX [Електронний ресурс]. - http://www.redov.ru/kompyutery\_i\_internet/osnovy\_programmirovanija\_v\_linux/p16.php
5. POSIX Threads Programming [Електронний ресурс]. - https://computing.llnl.gov/tutorials/pthreads/
6. Программирование многопоточных приложений. POSIX Threads. [Електронний ресурс]. - http://ssd.sscc.ru/ru/content/opplabs/loadbalancing

**ДОДАТКИ**

**Додаток А.**

**Лістинг програми ПРГ1**

//------------------------------------

//-- KR: part2 --

//-- Student: Skorichenko Oleksandr --

//-- Group: I0-22 --

//------------------------------------

//-- MА=max(E)\*(MO\*MK)+alfa\*MM --

//-- Data: 07.03.2015 --

//------------------------------------

#include "stdafx.h"

#include <iostream>

#include <vector>

#include <windows.h>

#include <omp.h>

#include <time.h>

using namespace std;

const int n = 500;

const int p = 4;

const int H = n / p;

int e = INT\_MIN;

int alfa;

vector<int> E(n);

vector<vector<int>> MA(n, vector<int>(n));

vector<vector<int>> MO(n, vector<int>(n));

vector<vector<int>> MK(n, vector<int>(n));

vector<vector<int>> MM(n, vector<int>(n));

vector<int> generateVector() {

vector<int> V(n);

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

V[i] = 1;

return V;

}

vector<vector<int>> generateMatrix() {

vector<vector<int>> MV(n, vector<int>(n));

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

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

MV[i][j] = 1;

return MV;

}

void outMatrix(vector<vector<int>> MV) {

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

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

cout << MV[i][j] << " ";

cout << endl;

}

cout << endl;

}

void main(void)

{

clock\_t t;

t = clock();

omp\_set\_dynamic(0);

omp\_set\_num\_threads(p);

#pragma omp parallel shared(e, MK, alfa)

{

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

printf("task T%i started\n", threadNumber + 1);

//Введення даних.

if (threadNumber + 1 == 1){

E = generateVector(); E[3] = 5;

MK = generateMatrix();

alfa = 3;

printf("task T1 input\n");

}

if (threadNumber + 1 == p){

MO = generateMatrix();

MM = generateMatrix();

printf("task T%i input\n", threadNumber + 1);

}

//Синхронізація по введенню даних.

#pragma omp barrier

int ei;

//Пошук максимуму.

ei = E[threadNumber\*H];

#pragma omp for

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

if (E[i] > ei)

ei = E[i];

#pragma omp critical(cs1)

{

if (ei > e)

e = ei;

}

//Синхронізація по обчисленню.

#pragma omp barrier

int alfai;

vector<vector<int>> MKi(n, vector<int>(n));

vector<vector<int>> MVi(n, vector<int>(n));

//Критична ділянка.

#pragma omp critical(cs2)

{

ei = e;

alfai = alfa;

MKi = MK;

}

//Обчислення.

#pragma omp for

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

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

for (int k = 0; k < n; k++){

MVi[i][j] += MO[i][k] \* MKi[k][j];

}

MA[i][j] = ei \* MVi[i][j] + alfai \* MM[i][j];

}

//Синхронізація по виведенню даних.

#pragma omp barrier

//Виведення даних.

if (threadNumber + 1 == 1){

Sleep(100);

printf("task T1: MA >>\n");

//outMatrix(MA);

//printf("%d\n", MA[0][0]);

}

printf("task T%i finished\n", threadNumber + 1);

}

t = clock() - t;

printf("time: %f seconds\n", ((double)t) / CLOCKS\_PER\_SEC);

system("pause");

}

**Додаток Б.**

**Лістинг програми ПРГ2**

Клас CourseWork(mpi).cpp:

/\*

------------------------------------

-- KR: part3 --

-- Student: Skorichenko Oleksandr --

-- Group: I0-22 --

------------------------------------

-- MА=max(E)\*(MO\*MK)+alfa\*MM --

-- Data: 22.04.2015 --

------------------------------------

\*/

#include "stdafx.h"

#include <mpi.h>

#include <iostream>

#include "Matrix.h"

#include <time.h>

using namespace std;

int N = 100;

int P = 10;

int H = N/P;

int FILLER = 1;

int first = 0;

int last = P/2 - 1;

void\* shiftPtrLeft(void\* inp, int s){

return (void\*)((int\*)inp + s\*H\*N);

}

void copyBuf(void\* from, void\* to, int size){

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

((int\*)to)[i] = ((int\*)from)[i];

}

}

bool isCentral(int rank){

if(rank == (P/4))

return true;

return false;

}

int findAndSendMinGlobal(int rank, int localMin){

int ei;

int bottomRank = rank + P/2;

MPI\_Status st;

if(rank != last){

MPI\_Recv(&ei, 1, MPI\_INT, rank+1, 21, MPI\_COMM\_WORLD, &st);

localMin = (ei < localMin) ? ei : localMin;

}

MPI\_Recv(&ei, 1, MPI\_INT, rank-1, 21, MPI\_COMM\_WORLD, &st);

localMin = (ei < localMin) ? ei : localMin;

MPI\_Recv(&ei, 1, MPI\_INT, bottomRank, 21, MPI\_COMM\_WORLD, &st);

localMin = (ei < localMin) ? ei : localMin;

MPI\_Request rqb, rql;

MPI\_Isend(&localMin, 1, MPI\_INT, bottomRank, 21, MPI\_COMM\_WORLD, &rqb);

MPI\_Isend(&localMin, 1, MPI\_INT, rank-1, 21, MPI\_COMM\_WORLD, &rql);

if(rank != last){

MPI\_Request rqr;

MPI\_Isend(&localMin, 1, MPI\_INT, rank+1, 21, MPI\_COMM\_WORLD, &rqr);

MPI\_Wait(&rqr, MPI\_STATUS\_IGNORE);

}

MPI\_Wait(&rqb, MPI\_STATUS\_IGNORE);

MPI\_Wait(&rql, MPI\_STATUS\_IGNORE);

return localMin;

}

int getMinGlobal(int rank, int localMin){

int globalMin;

if(isCentral(rank)){

globalMin = findAndSendMinGlobal(rank, localMin);

} else {

int ei;

MPI\_Status st;

if(rank < P/4){

MPI\_Recv(&ei, 1, MPI\_INT, rank + P/2, 21, MPI\_COMM\_WORLD, &st);

localMin = (ei < localMin) ? ei : localMin;

if(rank != first){

MPI\_Recv(&ei, 1, MPI\_INT, rank - 1, 21, MPI\_COMM\_WORLD, &st);

localMin = (ei < localMin) ? ei : localMin;

}

MPI\_Send(&localMin, 1, MPI\_INT, rank+1, 21, MPI\_COMM\_WORLD);

MPI\_Recv(&globalMin, 1, MPI\_INT, rank+1, 21, MPI\_COMM\_WORLD, &st);

if(rank != first){

MPI\_Send(&globalMin, 1, MPI\_INT, rank-1, 21, MPI\_COMM\_WORLD);

}

MPI\_Send(&globalMin, 1, MPI\_INT, rank+P/2, 21, MPI\_COMM\_WORLD);

} else {

MPI\_Recv(&ei, 1, MPI\_INT, rank + P/2, 21, MPI\_COMM\_WORLD, &st);

localMin = (ei < localMin) ? ei : localMin;

if(rank != last){

MPI\_Recv(&ei, 1, MPI\_INT, rank + 1, 21, MPI\_COMM\_WORLD, &st);

localMin = (ei < localMin) ? ei : localMin;

}

MPI\_Send(&localMin, 1, MPI\_INT, rank-1, 21, MPI\_COMM\_WORLD);

MPI\_Recv(&globalMin, 1, MPI\_INT, rank-1, 21, MPI\_COMM\_WORLD, &st);

if(rank != last){

MPI\_Send(&globalMin, 1, MPI\_INT, rank+1, 21, MPI\_COMM\_WORLD);

}

MPI\_Send(&globalMin, 1, MPI\_INT, rank+P/2, 21, MPI\_COMM\_WORLD);

}

}

return globalMin;

}

void calcMAh(Matrix& MA, int e, Matrix& MKh, int l, Matrix& ME, Matrix& MT, Matrix& MMh){

Matrix Res(N, H);

MT.mult(Res, MMh);

ME.mult(MA, Res);

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

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

MA[j][i] = e\*MKh[j][i] + l\*MA[j][i];

}

}

}

int size(Matrix& MA){

return MA.getColumns()\*MA.getRows();

}

void matrOut(Matrix& M){

for(int i = 0; i < M.getRows(); i++){

for(int j = 0; j < M.getColumns(); j++){

cout << M[i][j] << " ";

}

cout << endl;

}

}

void sendLeftMAh(int rank, Matrix& MAh){

Matrix MAh2(N, H);

MPI\_Recv(MAh2.getPtrToArray(), N\*H, MPI\_INT, rank + P/2, 31, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MAh.merge(MAh2);

if(rank != last){

Matrix MAh1(N, H\*(P - (rank+1)\*2));

MPI\_Recv(MAh1.getPtrToArray(), size(MAh1), MPI\_INT, rank+1, 31, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MAh.merge(MAh1);

}

MPI\_Send(MAh.getPtrToArray(), size(MAh), MPI\_INT, rank-1, 31, MPI\_COMM\_WORLD);

}

void threadFuncFirst(){

long transferStart1 = clock();

Matrix MO(N), ME(N);

MO.input();

ME.input();

int rightSS = N\*N - 2\*H\*N;

int rightRS = 2\*N\*H;

int fullSize = N\*N;

int bottomRank = 0 + P/2;

int flag = -1;

MPI\_Request req0;

MPI\_Isend(&flag, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, &req0);

MPI\_Request req4;

MPI\_Request req5;

MPI\_Isend(shiftPtrLeft(MO.getPtrToArray(), 2), rightSS, MPI\_INT, 1, 4, MPI\_COMM\_WORLD, &req4);

MPI\_Isend(ME.getPtrToArray(), fullSize, MPI\_INT, 1, 5, MPI\_COMM\_WORLD, &req5);

Matrix MKh(N,2\*H), MMh(N,2\*H);

Matrix MT(N);

int li;

MPI\_Status st1;

MPI\_Status st2;

MPI\_Status st3;

MPI\_Recv(&flag, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(MKh.getPtrToArray(), rightRS, MPI\_INT, 1, 1, MPI\_COMM\_WORLD, &st1);

MPI\_Recv(MMh.getPtrToArray(), rightRS, MPI\_INT, 1, 2, MPI\_COMM\_WORLD, &st2);

MPI\_Recv(MT.getPtrToArray(), fullSize, MPI\_INT, 1, 3, MPI\_COMM\_WORLD, &st3);

MPI\_Recv(&li, 1, MPI\_INT, 1, 6, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Wait(&req4, MPI\_STATUS\_IGNORE);

MPI\_Wait(&req5, MPI\_STATUS\_IGNORE);

MPI\_Request reql1, reql2, reql3, reql4, reql5, reql6;

MPI\_Isend(shiftPtrLeft(MKh.getPtrToArray(), 1), N\*H, MPI\_INT, bottomRank, 11, MPI\_COMM\_WORLD, &reql1);

MPI\_Isend(shiftPtrLeft(MMh.getPtrToArray(), 1), N\*H, MPI\_INT, bottomRank, 12, MPI\_COMM\_WORLD, &reql2);

MPI\_Isend(shiftPtrLeft(MO.getPtrToArray(), 1), N\*H, MPI\_INT, bottomRank, 14, MPI\_COMM\_WORLD, &reql4);

MPI\_Isend(MT.getPtrToArray(), N\*N, MPI\_INT, bottomRank, 13, MPI\_COMM\_WORLD, &reql3);

MPI\_Isend(ME.getPtrToArray(), N\*N, MPI\_INT, bottomRank, 15, MPI\_COMM\_WORLD, &reql5);

MPI\_Isend(&li, 1, MPI\_INT, bottomRank, 16, MPI\_COMM\_WORLD, &reql6);

MPI\_Wait(&reql1, MPI\_STATUS\_IGNORE);

MPI\_Wait(&reql2, MPI\_STATUS\_IGNORE);

MPI\_Wait(&reql3, MPI\_STATUS\_IGNORE);

MPI\_Wait(&reql4, MPI\_STATUS\_IGNORE);

MPI\_Wait(&reql5, MPI\_STATUS\_IGNORE);

MPI\_Wait(&reql6, MPI\_STATUS\_IGNORE);

MO.delExcessive(0, H);

MKh.delExcessive(0, H);

MMh.delExcessive(0, H);

long transferFinish1 = clock();

long cStart1 = clock();

int ei = MO.getMin();

long cFinish1 = clock();

int e = getMinGlobal(0, ei);

Matrix MAh(N, H);

long cStart2 = clock();

calcMAh(MAh, e, MKh, li, ME, MT, MMh);

long cFinish2 = clock();

long transferStart2 = clock();

Matrix MAh1(N, H);

MPI\_Recv(MAh1.getPtrToArray(), N\*H, MPI\_INT, bottomRank, 31, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MAh.merge(MAh1);

Matrix MAh2(N, N-2\*H);

MPI\_Recv(MAh2.getPtrToArray(), N\*(N-2\*H), MPI\_INT, 1, 31, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MAh.merge(MAh2);

long transferFinish2 = clock();

cout <<"Time to transfer: ";

cout << ((transferFinish1-transferStart1) + (transferFinish2 - transferStart2))/1000 << endl;

cout <<"Time to calculate1: ";

cout << (cFinish1 - cStart1)/1000 << endl;

cout <<"Time to calculate2: ";

cout << (cFinish2 - cStart2)/1000 << endl;

if(N < 13){

cout << "Matrix MA: \n";

for(int i = 0; i < MAh.getRows(); i++){

for(int j = 0; j < MAh.getColumns(); j++){

cout << MAh[i][j] << " ";

}

cout << endl;

}

}

}

void threadFuncLast(){

int bottomRank = P-1;

Matrix MK(N), MT(N), MM(N);

MK.input();

MT.input();

MM.input();

int li = FILLER;

int flag = 1;

MPI\_Request req0;

MPI\_Isend(&flag, 1, MPI\_INT, P/2-2, 0, MPI\_COMM\_WORLD, &req0);

MPI\_Request req1;

MPI\_Request req2;

MPI\_Request req3;

MPI\_Request req6;

MPI\_Isend(MK.getPtrToArray(), N\*N - 2\*H\*N, MPI\_INT, P/2-2, 1, MPI\_COMM\_WORLD, &req1);

MPI\_Isend(MM.getPtrToArray(), N\*N - 2\*H\*N, MPI\_INT, P/2-2, 2, MPI\_COMM\_WORLD, &req2);

MPI\_Isend(MT.getPtrToArray(), N\*N, MPI\_INT, P/2-2, 3, MPI\_COMM\_WORLD, &req3);

MPI\_Isend(&li, 1, MPI\_INT, P/2-2, 6, MPI\_COMM\_WORLD, &req6);

Matrix MOh(N, 2\*H), ME(N);

MPI\_Status st4;

MPI\_Status st5;

MPI\_Recv(&flag, 1, MPI\_INT, P/2-2, 0, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(MOh.getPtrToArray(), 2\*N\*H, MPI\_INT, P/2-2, 4, MPI\_COMM\_WORLD, &st4);

MPI\_Recv(ME.getPtrToArray(), N\*N, MPI\_INT, P/2-2, 5, MPI\_COMM\_WORLD, &st5);

MPI\_Wait(&req1, MPI\_STATUS\_IGNORE);

MPI\_Wait(&req2, MPI\_STATUS\_IGNORE);

MPI\_Wait(&req3, MPI\_STATUS\_IGNORE);

MPI\_Wait(&req6, MPI\_STATUS\_IGNORE);

MPI\_Request reql1, reql2, reql3, reql4, reql5/\*, reql6\*/;

MPI\_Isend(shiftPtrLeft(MK.getPtrToArray(), P - 1 ), N\*H, MPI\_INT, bottomRank, 11, MPI\_COMM\_WORLD, &reql1);

MPI\_Isend(shiftPtrLeft(MM.getPtrToArray(), 1), N\*H, MPI\_INT, bottomRank, 12, MPI\_COMM\_WORLD, &reql2);

MPI\_Isend(shiftPtrLeft(MOh.getPtrToArray(), 1), N\*H, MPI\_INT, bottomRank, 14, MPI\_COMM\_WORLD, &reql4);

MPI\_Isend(MT.getPtrToArray(), N\*N, MPI\_INT, bottomRank, 13, MPI\_COMM\_WORLD, &reql3);

MPI\_Isend(ME.getPtrToArray(), N\*N, MPI\_INT, bottomRank, 15, MPI\_COMM\_WORLD, &reql5);

MPI\_Isend(&li, 1, MPI\_INT, bottomRank, 16, MPI\_COMM\_WORLD, &reql5);

MPI\_Wait(&reql1, MPI\_STATUS\_IGNORE);

MPI\_Wait(&reql2, MPI\_STATUS\_IGNORE);

MPI\_Wait(&reql3, MPI\_STATUS\_IGNORE);

MPI\_Wait(&reql4, MPI\_STATUS\_IGNORE);

MPI\_Wait(&reql5, MPI\_STATUS\_IGNORE);

MK.delExcessive((P-2)\*H, (P-1)\*H);

MM.delExcessive((P-2)\*H, (P-1)\*H);

MOh.delExcessive(0, H);

int ei = MOh.getMin(0, H);

int e = getMinGlobal(last, ei);

Matrix MAh(N, H);

calcMAh(MAh, e, MK, li, ME, MT, MM);

sendLeftMAh(last, MAh);

}

void threadFuncMed(int rank){

int bottomRank = rank + P/2;

int rightSS = (P - rank\*2 - 2)\*H\*N;

int rightSR = (rank\*2 + 2)\*H\*N;

int leftSS = rank\*H\*2\*N;

int leftSR = (P-2\*rank)\*H\*N;

int flag;

MPI\_Request rq[7];

MPI\_Request rqf;

MPI\_Recv(&flag, 1, MPI\_INT, MPI\_ANY\_SOURCE, 0, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

Matrix MKh(N, rightSR/N), MMh(N, rightSR/N), MT(N);

int li;

Matrix MOh(N, leftSR/N), ME(N);

if(flag == 1){

MPI\_Recv(MKh.getPtrToArray(), rightSR, MPI\_INT, rank+1, 1, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(MMh.getPtrToArray(), rightSR, MPI\_INT, rank+1, 2, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(MT.getPtrToArray(), size(MT), MPI\_INT, rank+1, 3, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(&li, 1, MPI\_INT, rank+1, 6, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Isend(&flag, 1, MPI\_INT, rank-1, 0, MPI\_COMM\_WORLD, &rq[0]);

MPI\_Isend(MKh.getPtrToArray(), leftSS, MPI\_INT, rank-1, 1, MPI\_COMM\_WORLD, &rq[1]);

MPI\_Isend(MMh.getPtrToArray(), leftSS, MPI\_INT, rank-1, 2, MPI\_COMM\_WORLD, &rq[2]);

MPI\_Isend(MT.getPtrToArray(), size(MT), MPI\_INT, rank-1, 3, MPI\_COMM\_WORLD, &rq[3]);

MPI\_Isend(&li, 1, MPI\_INT, rank-1, 6, MPI\_COMM\_WORLD, &rq[6]);

MPI\_Recv(&flag, 1, MPI\_INT, MPI\_ANY\_SOURCE, 0, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(MOh.getPtrToArray(), leftSR, MPI\_INT, rank-1, 4, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(ME.getPtrToArray(), size(ME), MPI\_INT, rank-1, 5, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Isend(&flag, 1, MPI\_INT, rank+1, 0, MPI\_COMM\_WORLD, &rqf);

MPI\_Isend(shiftPtrLeft(MOh.getPtrToArray(), 2), rightSS, MPI\_INT, rank+1, 4, MPI\_COMM\_WORLD, &rq[4]);

MPI\_Isend(ME.getPtrToArray(), size(ME), MPI\_INT, rank+1, 5, MPI\_COMM\_WORLD, &rq[5]);

} else {

MPI\_Recv(MOh.getPtrToArray(), leftSR, MPI\_INT, rank-1, 4, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(ME.getPtrToArray(), size(ME), MPI\_INT, rank-1, 5, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Isend(&flag, 1, MPI\_INT, rank+1, 0, MPI\_COMM\_WORLD, &rqf);

MPI\_Isend(shiftPtrLeft(MOh.getPtrToArray(), 2), rightSS, MPI\_INT, rank+1, 4, MPI\_COMM\_WORLD, &rq[4]);

MPI\_Isend(ME.getPtrToArray(), size(ME), MPI\_INT, rank+1, 5, MPI\_COMM\_WORLD, &rq[5]);

MPI\_Recv(&flag, 1, MPI\_INT, MPI\_ANY\_SOURCE, 0, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(MKh.getPtrToArray(), rightSR, MPI\_INT, rank+1, 1, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(MMh.getPtrToArray(), rightSR, MPI\_INT, rank+1, 2, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(MT.getPtrToArray(), size(MT), MPI\_INT, rank+1, 3, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Recv(&li, 1, MPI\_INT, rank+1, 6, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

MPI\_Isend(&flag, 1, MPI\_INT, rank-1, 0, MPI\_COMM\_WORLD, &rq[0]);

MPI\_Isend(MKh.getPtrToArray(), leftSS, MPI\_INT, rank-1, 1, MPI\_COMM\_WORLD, &rq[1]);

MPI\_Isend(MMh.getPtrToArray(), leftSS, MPI\_INT, rank-1, 2, MPI\_COMM\_WORLD, &rq[2]);

MPI\_Isend(MT.getPtrToArray(), size(MT), MPI\_INT, rank-1, 3, MPI\_COMM\_WORLD, &rq[3]);

MPI\_Isend(&li, 1, MPI\_INT, rank-1, 6, MPI\_COMM\_WORLD, &rq[6]);

}

for(int i = 0; i < 7; i++){

MPI\_Wait(&rq[i], MPI\_STATUSES\_IGNORE);

}

MPI\_Wait(&rqf, MPI\_STATUSES\_IGNORE);

MKh.delExcessive(size(MKh)/N-2\*H, size(MKh)/N);

MMh.delExcessive(size(MMh)/N-2\*H, size(MMh)/N);

MOh.delExcessive(0, 2\*H);

MPI\_Request rq1[6];

MPI\_Isend(shiftPtrLeft(MKh.getPtrToArray(), 1), N\*H, MPI\_INT, bottomRank, 11, MPI\_COMM\_WORLD, &rq1[0]);

MPI\_Isend(shiftPtrLeft(MMh.getPtrToArray(), 1), size(MMh) - N\*H, MPI\_INT, bottomRank, 12, MPI\_COMM\_WORLD, &rq1[1]);

MPI\_Isend(MT.getPtrToArray(), size(MT), MPI\_INT, bottomRank, 13, MPI\_COMM\_WORLD, &rq1[2]);

MPI\_Isend(shiftPtrLeft(MOh.getPtrToArray(), 1), size(MOh) - N\*H, MPI\_INT, bottomRank, 14, MPI\_COMM\_WORLD, &rq1[3]);

MPI\_Isend(ME.getPtrToArray(), size(ME), MPI\_INT, bottomRank, 15, MPI\_COMM\_WORLD, &rq1[4]);

MPI\_Isend(&li, 1, MPI\_INT, bottomRank, 16, MPI\_COMM\_WORLD, &rq1[5]);

for(int i = 0; i < 6; i++){

MPI\_Wait(&rq1[i], MPI\_STATUSES\_IGNORE);

}

MKh.delExcessive(0, H);

MMh.delExcessive(0, H);

MOh.delExcessive(0, H);

int ei = MOh.getMin(0, H);

int e = getMinGlobal(rank, ei);

Matrix MAh(N, H);

calcMAh(MAh, e, MKh, li, ME, MT, MMh);

sendLeftMAh(rank, MAh);

}

void threadFuncBottom(int rank){

Matrix MOh(N, H), MKh(N, H), MMh(N, H), MT(N), ME(N);

int li;

MPI\_Status st1;

MPI\_Status st2;

MPI\_Status st3;

MPI\_Status st4;

MPI\_Status st5;

MPI\_Status st6;

MPI\_Recv(MKh.getPtrToArray(), N\*H, MPI\_INT, rank - P/2, 11, MPI\_COMM\_WORLD, &st1);

MPI\_Recv(MMh.getPtrToArray(), N\*H, MPI\_INT, rank - P/2, 12, MPI\_COMM\_WORLD, &st2);

MPI\_Recv(MT.getPtrToArray(), N\*N, MPI\_INT, rank - P/2, 13, MPI\_COMM\_WORLD, &st3);

MPI\_Recv(MOh.getPtrToArray(), N\*H, MPI\_INT, rank - P/2, 14, MPI\_COMM\_WORLD, &st4);

MPI\_Recv(ME.getPtrToArray(), N\*N, MPI\_INT, rank - P/2, 15, MPI\_COMM\_WORLD, &st5);

MPI\_Recv(&li, 1, MPI\_INT, rank - P/2, 16, MPI\_COMM\_WORLD, &st6);

int ei = MOh.getMin();

MPI\_Send(&ei, 1, MPI\_INT, rank - P/2, 21, MPI\_COMM\_WORLD);

int e;

MPI\_Recv(&e, 1, MPI\_INT, rank - P/2, 21, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

Matrix MAh(N, H);

calcMAh(MAh, e, MKh, li, ME, MT, MMh);

MPI\_Send(MAh.getPtrToArray(), N\*H, MPI\_INT, rank - P/2, 31, MPI\_COMM\_WORLD);

}

int main(int argc, char\* argv[])

{

MPI\_Init(&argc, &argv);

long tStart = clock();

int rank;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &P);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

H = N/P;

last = P/2 - 1;

cout << "Thread " << rank << " started.\n";

if(rank == first){

threadFuncFirst();

} else {

if(rank == last){

threadFuncLast();

} else {

if(rank > (last)){

threadFuncBottom(rank);

} else {

threadFuncMed(rank);

}

}

}

cout << "Thread " << rank << " finish.\n";

MPI\_Finalize();

if(rank == first){

long tFinish = clock();

cout << endl << "time: " << (tFinish - tStart)/1000 << "s " << (tFinish - tStart)%1000 << "ms" << endl;

char ch;

cin >> ch;

}

return 0;

}

Клас Matrix.h:

#pragma once

#include <iostream>

class Matrix {

private:

static const int FILLER = 1;

//int size;

int rows;

int columns;

public:

int \*matrix;

Matrix(int size);

Matrix(int rows, int columns);

~Matrix();

int getRows();

int getColumns();

void\* getPtrToArray();//--

int\* operator [] (int row);

void input();

void copy(Matrix& copyMatrix);

void mult(Matrix& res, Matrix& multMatr);

int getMin();

int getMin(int from, int to);

void delExcessive(int from, int to);

void merge(Matrix& MAh);

};

std::ostream& operator << (std::ostream &out, Matrix &matrix);

Клас Matrix.cpp:

#include "Matrix.h"

#include <iostream>

Matrix::Matrix(int size) {

this->rows = size;

this->columns = size;

matrix = new int [size\*size];

}

Matrix::Matrix(int rows, int columns) {

this->rows = rows;

this->columns = columns;

matrix = new int [rows\*columns];

}

Matrix::~Matrix() {

delete(matrix);

}

int Matrix::getRows() {

return rows;

}

int Matrix::getColumns() {

return columns;

}

void\* Matrix::getPtrToArray(){

return (void\*)(&(\*matrix));

}

int\* Matrix::operator[](int row) {

return matrix + row\*columns;

}

void Matrix::input() {

for ( int i = 0; i < rows\*columns; i++ ) {

matrix[i] = FILLER;

}

}

void Matrix::copy(Matrix& copyMatrix) {

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

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

copyMatrix[i][j] = matrix[i\*columns + j];

}

}

}

void Matrix::mult(Matrix& res, Matrix& multMatr){

int\* r;

int\* mm;

int\* tm;

int rColumns = res.getColumns();

int mmColumns = multMatr.getColumns();

int tmColumns = columns;

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

{

r = res.matrix;

tm = matrix;

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

{

r[i] = 0;

mm = multMatr.matrix;

for (int k = 0; k < rows; k++)

{

r[i] += tm[k] \* mm[i];

mm += mmColumns;

};

r += rColumns;

tm += tmColumns;

}

}

}

int Matrix::getMin(){

int min = matrix[0];

for(int i = 1; i < rows\*columns; i++){

if(min > matrix[i]){

min = matrix[i];

}

}

return min;

}

int Matrix::getMin(int from, int to){

int min = matrix[from];

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

for(int j = from; j < to; j++){

if(min > matrix[i]){

min = matrix[i];

}

}

}

return min;

}

void Matrix::delExcessive(int from, int to){

int\* accum = new int[(to-from)\*rows];

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

for(int j = from; j < to; j++){

accum[i\*(to-from) - from + j] = matrix[i\*columns + j];

}

}

int\* toDel = matrix;

matrix = accum;

delete[] toDel;

columns = to-from;

}

void Matrix::merge(Matrix& MAh){

int size1 = columns\*rows;

int size2 = MAh.getColumns()\*MAh.getRows();

int\* accum = new int[size1 + size2];

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

for(int j = 0; j < (columns + MAh.getColumns()); j++){

if(j < columns){

accum[i\*(columns + MAh.getColumns()) + j] = matrix[i\*columns + j];

} else {

accum[i\*(columns + MAh.getColumns()) + j] = MAh[i][j-columns];

}

}

}

int\* toDel = matrix;

matrix = accum;

delete[] toDel;

columns += MAh.getColumns();

}

std::ostream& operator<<(std::ostream &out, Matrix &matrix) {

int last = matrix.getColumns() - 1;

out << "Matrix:" << std::endl;

for (int row = 0; row < matrix.getRows(); row++) {

for (int col = 0; col < last; col++)

out << matrix[row][col] << " ";

out << matrix[row][last] << std::endl;

}

return out;

};



Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

1

ІАЛЦ 462637.003

Розроб.

Скоріченко О.В.

Перевір.

Корочкін О.В.

Реценз.

Н. Контр.

Затверд.

Додаток В. Блок-схема алгоритму головної процедури ПРГ1

Літ.

Акрушів

1

НТУУ «КПІ»

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

1

ІАЛЦ 462637.003

Розроб.

Скоріченко О.В.

Перевір.

Корочкін О.В.

Реценз.

Н. Контр.

Затверд.

Додаток Г. Алгоритм головної програми ПРГ1

Літ.

Акрушів

1

НТУУ «КПІ»

