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 (ки) 3 курсу \_\_\_ІО-22\_\_\_ групи

напряму підготовки 050102 «Комп’ютерна інженерія»

Сергін Д.І

(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

(прізвище, ім’я, по батькові)

Сергін Дмитро Ігорович

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 2015 р.

3. Вхідні дані до роботи

- процеси в сучасних бібліотеках паралельного програмування

- Ада.Захищені модулі..



- MPI. .



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

- огляд процесів в сучасних бібліотеках паралельного програмування

- розробка і тестування програми ПРГ1 для ПКС CП

- розробка і тестування програми ПРГ2 для ПКС ЛП

5. Перелік графічного матеріалу

- структурна схема ПКС CП

- 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 грудня 2015 р.

#### 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 \_\_\_\_\_\_\_\_\_** \_\_\_\_\_\_\_\_\_ Сергін Д.І \_\_\_\_\_\_

( підпис ) (прізвище та ініціали)

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

( підпис ) (прізвище та ініціали)

**ЗМІСТ**

ВСТУП…………………………………………………………………………… 5

РОЗДІЛ 1.ОГЛЯД ПРОЦЕСІВ В СУЧАСНИХ БІБЛІОТЕКАХ ПРОГРАМУВАННЯ ………………………………………………………………………………..…... 6

* 1. Процеси в бібліотеці OpenMP………………………………………. 6
  2. Процеси в бібліотеці MPI…………………………………………….9
  3. Процеси в бібліотеці PVM…………………………………………...12
  4. Процеси в бібліотеці WinAPI………………………………………..15
  5. Процеси в бібліотеці POSIX Threads………………………………..19
  6. Висновки по розділу 1………………………………………………...20

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС ОП…………………20

* 1. Розробка паралельного математичного алгоритму……………… 20
  2. Розробка алгоритмів процесів……………………………………… 22
  3. Розробка схеми взаємодії процесів…………………………………24
  4. Розробка програми ПРГ1…………………………………………… 26
  5. Тестування програми ПРГ1 …………………………………………27
  6. Висновки до розділу 2 …………………………………………… 29

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП ……..………… 30

* 1. Розробка паралельного математичного алгоритму……………… 30
  2. Розробка алгоритмів процесів…………………………………… 32
  3. Розробка схеми взаємодії процесів………………………… … 34
  4. Розробка програми ПРГ2……………………………………… 36
  5. Тестування програми ПРГ2……………………………………… 37
  6. Висновки до розділу 3………………………………………………39

ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ДО РОБОТИ………………………40

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……………………………………. 43

ДОДАТКИ……………………………………………………............................. 45

**ВСТУП**

Паралельні і розподілені обчислення дозволяють вирішувати складні та ресурсоємкі задачі, які доволі важко вирішувати звичайними обчислювальними засобами.

Паралельні обчислення вимагають ретельної розробки паралельного математичного алгоритму та кожного кроку його реалізації , тому для вирішення задачі необхідно провести ряд досліджень, які будуть виконані в курсовій роботі.

В курсовій роботі розглядаються сучасні засоби програмування для паралельних комп’ютерних систем. Виконується огляд паралельних комп’ютерних систем із спільною та локальною пам’яттю , розробляються дві програми для них, обчислюються коефіцієнти ефективності та прискорення.

В розділі 1 виконується огляд процесів в сучасних бібліотеках паралельного програмування, розглядаються засоби створення, конфігурації та взаємодії паралельних потоків ,оцінюються переваги та недоліки кожної з бібліотек .

В розділі 2 виконується розробка паралельного математичного алгоритму та написання програми для вирішення математичної задачі. Програма створюється на мові програмування Ada з використанням механізму захищених модулів. Кількість паралельних задач-8. Програма створюється для паралельної комп’ютерної системи зі спільною пам’яттю. Виконується оцінка ефективності написаної програми при змінній кількості ядер процесора в системі.

В розділі 3 виконується розробка математичного алгоритму та написання програми для вирішення математичної задачі в паралельній системі з локальною пам’яттю. Оцінюється швидкодія та ефективність створеної програми.

# РОЗДІЛ 1. ОГЛЯД ПРОЦЕСІВ В СУЧАСНИХ БІБЛІОТЕКАХ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ

Процес – опис певних дій, що відбуваються при виконанні програми . Кожен процес має власну, виділену для себе, ділянку памяті та певний набір ресурсів.

В паралельному програмуванні розглядаються паралельні процеси- ті процеси, які виконуються в паралельних програмах . Для створення паралельних процесів створено бібліотеки: POSIX Threads, MPI, PVM, OpenMP, WinAPI. Процеси можуть бути створені за допомогою потокових функцій (у бібліотеках WinAPI, POSIX Threads ) та шляхом створення копії ділянки паралельної програми (PVM, OpenMP, MPI).

Важливою задачею при програмуванні паралельних комп’ютерних програм є взаємодія процесів. Паралельне виконання дій в програмі може призвести до помилок при звертанні до спільних ресурсів, тому процеси мають взаємодіяти між собою при доступі до них. Для вирішення програміст має вирішувати задачі взаємного виключення та синхронізації. Паралельні бібліотеки мають широкий набір засобів для організації правильної роботи паралельних процесів, такі як : м’ютекси, семафори, критичні ділянки, монітори, захищені модулі, розподілені змінні, механізми посилки повідомлень між процесами.

* 1. **Процеси в бібліотеці OpenMP**

OpenMP (Open Multi-Processing) - відкрита бібліотека для створення паралельних програм [1]. Бібліотека дає опис сукупності директив компілятора, бібліотечних процедур , що призначені для програмування багатопоточних програм на багатопроцесорних системах із розподільною пам’яттю.

Перша версія бібліотеки OpenMP була створена в 1997 для мови програмування Fortran, для C++ версія бібліотеки реалізована в 1998 році, в 2008 році створена остання версія OpenMP 3.0, яку можна використовувати в багатьох мовах програмування.

Розробники поєднали в бібліотеці можливості створення потоків в програмах, не вимагаючи, при цьому, від програміста знань про створення, синхронізацію та знищення потоків, а також необхідності точного визначення кількості необхідних потоків. Для реалізації даних особливостей бібліотека має набір прагм, директив,функцій та змінних середовища, які явно вказують компілятору , як та коли створити та керувати процесом в програмі.

Головною особливістю OpenMP є те, що програма початково є послідовною, а паралелізм в ній забезпечується за допомогою паралельного виконання послідовних частин.

Робота OpenMP- програми починається з одного процесу- основного(Master Thread ). В програмі можуть створюватись паралельні секції , при вході в які, основний потік створює групи паралельних потоків. В кінці паралельної ділянки групи потоків блокуються, а виконання Master потоку продовжується. В паралельній ділянці може міститись інша паралельна ділянка(вкладена),в якій кожен потік початкової ділянки стає основним для своєї групи потоків.

Бібліотека проста в користуванні і використовує два типи базових конструкцій:

* Директиви pragma.
* Функції виконуючого середовища.

Директива має загальний вигляд : #pragma omp <директива> -

де, #pragma omp–є префіксом, а директива-вказівка компілятору на виконання паралельного блоку коду.

OpenMP підтримує директиви parallel, for, section, sections, single, master, critical, flush, ordered та atomic, які визначають механізми виконання чи конструкції синхронізації [2]. За допомогою директиви parallel визначається паралельна ділянка коду, при виконанні тіла прагми створються копії її тіла, кожна з яких виконується в окремому потоці , кількість яких встановлюється системою чи користувачем. Також кожну прагму #pragma omp parallel можна розбити на секції ,що можуть виконуватись в окремих потоках.

Функції OpenMP слугують в основному для зміни та отримання параметрів середовища та підтримки деяких типів синхронізації.

Головним механізмом для застосування паралелізму є процес копіювання. Кожна копія має свій унікальний ідентифікатор, необхідний для доступу до певного процесу, головний процес завжди має ідентифікатор -0. Кількість копій визначається за допомогою функції set\_num\_threads(n),де n-кількість копій.

Однією з найбільш корисних можливостей бібліотеки є можливість створення паралельних циклів за допомогою директиви #pragma omp for ,яка дозволяє виконувати частини тіла циклу в окремих процесах.

#pragma omp parallel // створення паралельної ділянки

{

#pragma omp for / /директива про паралельне виконання циклу

for(int i = 1; i < size; ++i) // створення циклу

x[i] = (y[i-1] + y[i+1])/2; // тіло циклу

}

В наведеному коді директива #pragma omp for повідомляє, що при виконанні цикла в паралельній ділянці ітерації будуть розділені між потоками групи.

void QuickSort (int numList[], int nLower, int nUpper)

{

if (nLower < nUpper)

{

// Розбиття інтервалу сортування

int nSplit = Partition (numList, nLower, nUpper);

#pragma omp parallel sections // паралельні секції

{

#pragma omp section // конкретна секція

QuickSort (numList, nLower, nSplit - 1);

#pragma omp section

QuickSort (numList, nSplit + 1, nUpper);

}

}

}

Вище наведено приклад використання паралельних секцій. Директива #pragma omp parallel sections створює паралельні ділянку з секціями. Кожній секції в паралельній ділянці ставиться у відповідність один з групи потоків і всі секції виконуються одночасно.

Для організації взаємодії процесів в бібліотеці використовуються відповідні прагми: omp atomic, omp critical –для вирішення задач взаємного виключення, omp barrier – для вирішення задачі синхронізації. Прагма omp critical використовується для описання програмних блоків, які будуть виконуватись тільки в одному потоці.

Прагма omp atomic забезпечує запобігання переривання доступу ,зчитування та запису даних , які знаходяться в спільній памяті зі сторони інших потоків, даний вид організації значно збільшує час виконання програми. Прагма omp barrier встановлює режим очікування завершення роботи усіх запущених в програмі потоків, даний вид синхронізації присутній неявно , як і omp atomic , в прагмі omp atomic, для вимкнення неявної синхронізації використовується директива nowait.

**1.2 Процеси в бібліотеці MPI**

MPI(Message Passing Interface)- одна з найбільш поширених бібліотек для створення паралельних процесів в програмах для комп’ютерних систем з розподіленою пам’яттю.

Першу версію бібліотеки розроблено в 1994 році під назвою MPI 1.0, розробка та підтримка бібліотеки продовжується, останньою версією є MPI 3.0, яка була реалізована в 2012 році.

MPI – бібліотека функцій , призначених підтримки роботи паралельних процесів з використанням механізму обміну повідомленнями між процесами.

Процеси в MPI мають назву задач. Всі задачі об’єднуються в групи- комунікатори для полегшення доступу до кожної задачі, оскільки комунікатор може визначати дії лише для задач групи, звертатись до групи , дозволяє відправляти повідомлення задачам групи.

Для створення паралельних процесів бібліотека має базові функції:

* **int MPI\_Init( int\* argc, char\*\*\* argv) – виконує ініціалізацію паралельної частини програми. Ініціалізація виконується для кожної програми лише один раз. Всі функції бібліотеки можуть бути викликані лише після виклику даної функції. У разі правильного виконання повертає** MPI\_SUCCESS, в іншому випадку- код помилки.
* **int MPI\_Finalize( void )- завершення паралельної частини програми, всі наступні виклики процедур бібліотеки заборонені. У момент виклику функції MPI\_Finalize всі дії процесу, які потребують його участі в обміні повідомленнями мають бути завершені.**
* **int MPI\_Comm\_rank( MPI\_Comm comm, int\* rank)- функція, яка визначає номер процесу(ідентифікатор ) в групі. Ранк має межі від 0** доsize\_of\_group-1*.* Кожен процес має свій унікальний ідентифікатор.
* **int MPI\_Comm\_size( MPI\_Comm comm, int\* size)- функція, яка дозволяє встановити кількість паралельних процесів в комунікаторі(групі). Повертає розмір групи.**

**Кожна функція чи команда бібліотеки має розпочинатись з префікса mpi.**

При створенні паралельної ділянки спочатку всі задачі належать до одного базового комунікатору, який має назву mpi\_Comm\_World, який створюється автоматично при виклику функції **MPI\_Init. Користувач має можливість створення власних груп за допомогою функції mpi\_Group.**

**Паралельність процесів забезпечується одночасним виконанням задач комунікатора, які являють собою копії заданої програми.**

Приклад програми на мові програмування Х з використання бібліотеки програмування MPI:

#include <stdio.h>

#include "mpi.h"

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

{

int rank, size;//змінні для отримання розміру групи та номера процесу

MPI\_Init(&argc, &argv);// ініціалізація паралельної ділянки

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);//визначення кількості паралельних процесів

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);// визначення номеру процесу

printf ("process %d, size %d\n", rank, size);//виведення розміру групи та номера процесу

MPI\_Finalize();//завершення паралельної ділянки

}

Базовим механізмом зв’язку між процесами є передача та прийом повідомлень [2]. Процеси можуть по-різному обмінюватись інформацією між собою: один процес може посилати іншому процесу повідомлення , процес може використати комунікатор для передачі повідомлення групі процесів, також можливий синхронний варіант обміну повідомлень, за якого процес при відправленні повідомлення блокується доти, поки інший процес не прийме повідомлення і навпаки , та асинхронний варіант, за якого процес відправляє повідомлення, але не чекає його отримання.

Для прийому та передачі повідомлень з блокуванням процесів використовуються такі функції [4] :

* **int MPI\_Send(void\* buf, int count, MPI\_Datatype datatype, int dest, int msgtag, MPI\_Comm comm)- виконує посилання повідомлення з ідентифікатором mstag, з count елементів типу datatype, процесу з номером dest, всі елементи повідомлення знаходяться в буфері buf . Дозволяє відправляти повідомлення самому собі.**
* **int MPI\_Recv(void\* buf, int count, MPI\_Datatype datatype, int source, int msgtag, MPI\_Comm comm, MPI\_Status \*status)- відповідає за прийом повідомлення з ідентифікатором** msgtagвід процесу source з блокуванням, число елементів повідомлення не повинно перевищувати count, в якості процеса відправника можна вказати константу MPI\_ANY\_TAG –ознаку того, що повідомлення міг відправити будь-який процес.

Для прийому та передачі повідомлень без блокування можуть використовуватись такі функції :

* **int MPI\_Isend(void \*buf, int count, MPI\_Datatype datatype, int dest, int msgtag, MPI\_Comm comm, MPI\_Request \*request)-виконує відправку повідомлення, аналогічну** MPI\_Send, але повернення з підпрограми виконується одразу після ініціалізації процесу передачі повідомлення без очікування обробки всього повідомлення, яке знаходиться в буфері.
* **int MPI\_Irecv(void \*buf, int count, MPI\_Datatype datatype, int source, int msgtag, MPI\_Comm comm, MPI\_Request \*request)- прийом повідомлення, аналогічний** MPI\_Recv, але повернення з підпрограми виконується одразу після прийому без очікування отримання повідомлення в буфері.
* **int MPI\_Wait( MPI\_Request \*request, MPI\_Status \*status)-очікування завершення асинхронних процедур** MPI\_Isendчи MPI\_Irecv*,* прив’язаних до ідентифікатора request.

Бібліотека MPI є одним з найефективніших інструментів паралельного програмування, практично для будь-якої комп’ютерної системи існує відповідна реалізація даної бібліотеки, тому не потрібно вирішувати проблеми переносу програмного забезпечення між комп’ютерними системами.

**1.3 Процеси в бібліотеці PVM**

PVM (Parallel Virtual Machine)[5] - набір програмних компонентів, який дозволяє створювати паралельну комп’ютерну систему. Бібліотека була створена в 1989 році для вирішення складних задач паралельного програмування. PVM працює як потужна віртуальна машина створена на основі мережі комп’ютерів чи серверів з розподіленою пам’яттю . При необхідності віртуальна машина може використовувати можливості окремих комп’ютерів з мережі для виконання задач без потреби в суперкомпютері.

Бібліотека реалізована для використання у мовах програмування С, С++ , Fortran.

Для підключення паралельної ділянки необхідно викликати функцію pvm\_start\_pvmd( int argc, char \*\*argv, int block ).

Породження паралельних процесів відбувається за допомогою функції pvm\_spawn( char \*task, char \*\*argv, int flag, char \*where, int ntask, int \*tids ) , в якості параметрів передаються назва процесу(виконавчого файлу), прапорець, з опціями породження, параметр, що визначає місце початку паралельного процесу(в залежності від прапорця, цей параметр може бути іменем хосту чи класу PVM архітектури), число копій програми, масив ідентифікаторів створюваних процесів, кількість створюваних процесів.

Для завершення певного процесу необхідно викликати функцію pvm\_kill(). Ідентифікатор певної задачі отримуються при використанні функцій pvm\_mytid(), pvm\_gettid(), які повертають ідентифікатор процесу чи групи процесів.

Процеси в бібліотеці PVM можуть взаємодіяти між собою за допомогою механізму пересилки повідомлень. Функція pvm\_send() виконує передачу повідомлення з активного буфера, а для розсилки повідомлень між процесами, що знаходяться в групі використовується функція pvm\_mcast().

Приклад програми :

#include "pvm3.h"

#include <stdio.h>

#define SIZE 1000

#define NPROCS 5

main()

{

   int mytid, task\_ids[NPROCS];//кількість процесів

   int a[SIZE], results[NPROCS], sum = 0;

   int i, msgtype, num\_data = SIZE/NPROCS;

   mytid = pvm\_mytid();//отримання ідентифікатора

   for (i = 0; i < SIZE; i++)

      a[i] = i % 25;

   pvm\_spawn("worker", (char \*\*)0, PvmTaskDefault, "", NPROCS, task\_ids);//створення процесів

   for (i = 0; i < NPROCS; i++) {

      pvm\_initsend(PvmDataDefault);//створення буфера передачі

      pvm\_pkint(&num\_data, 1, 1);

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

      pvm\_send(task\_ids[i], 4);//передача повідомлення для робочих процесів

   }

   msgtype = 7;//чекаємо і збираємо результати

   for (i = 0; i < NPROCS; i++) {

      pvm\_recv(task\_ids[i], msgtype);//прийом результату

      pvm\_upkint(&results[i], 1, 1);//розпакування результату

      sum += results[i];

   }

   printf("The sum is %d \n",sum);

   pvm\_exit();

}

Взаємодія процесів в PVM відбувається за допомогою механізму обміну повідомленнями. Відправлення повідомлення в бібліотеці відбувається в три етапи.

По-перше, відбувається ініціалізація буферу передачі шляхом виклику функцій pvm\_initsend() чи pvm\_mkbuf(). По-друге, повідомлення має бути упаковане в цей буфер за допомогою довільної кількості викликів підпрограм pvm\_pk\*() в будь-якій комбінації. Далі підготовлене повідомлення відправляється відповідному процесу шляхом виклику pvm\_send()чи за допомогою процедури pvm\_mcast().

Прийом повідомлення відбувається за виклику функції блокуючого чи неблокуючого прийому , далі кожен з упакованих фрагментів розпаковується в буфер прийому. Функції прийому можуть бути налаштовані на прийом будь-якого повідомлення , функція pvm\_recvf() дозволяє користувачам визначати свої власні контексти прийому в яких будуть працювати всі наступні функції прийому PVM.

Докладніше про функції прийому та передачі повідомлень:

* int info = pvm\_send( int tid, int msgtag)- функція помічає повідомлення цілочисленним ідентифікатором та передає його безпосередньо процесу з ідентифікатором tid.
* int info = pvm\_mcast( int \*tids, int ntask, int msgtag)- функція помічає повідомлення цілочисленним ідентифікатором msgtag та передає його всім задачам, вказаним в цілочисленному масиві ідентифікаторів tids, який має довжину ntask.
* int bufid = pvm\_recv( int tid, int msgtag)- функція блокуючого прийому буде очікувати до моменту , поки від задачі з ідентифікатором tid не прийде повідомлення з міткою msgtag. Після надходження вона розміщує повідомлення в новий активний буфер прийому. Попередній активний буфер прийому очищується, якщо він був збережений.
* int bufid = pvm\_nrecv(int tid, int msgtag)- якщо запрошене повідомлення не прибуло, то функція неблокуючого прийому поверне 0. Ця підпрограма може викликатись скільки завгодно разів для певного повідомлення з ціллю перевірки його прибуття. Якщо прибуло повідомлення з міткою msgtag, функція розміщує це повідомлення в новий створений буфер і повертає ідентифікатор буфера.

PVM є зручним інструментом для створення паралельних комп’ютерних систем . Бібліотека дозволяє створити віртуальну машину з мережі комп’ютерів і розбивати будь-яку задачу між ними при одночасному виконанні.

**1.4 Процеси в бібліотеці WinAPI**

WinAPI(application programming interfaces) – бібліотека функцій, розроблена для OС Windows. У WinAPI реалізовано набір функцій, необхідних для паралельного програмування.

Бібліотека використовується при програмуванні на мовах С, С++ та подібних до них. Найбільш популярною версією в наш час є Win32, яка розроблена для 32-розрядних операційних систем. Бібліотечні функції використовуються для створення та знищення потоків, синхронізації доступу до роздільних змінних, описують моделі для програмування координації дій легких процесів.

Для створення потоку необхідно використовувати змінні типу HANDLE, за допомогою яких можна звертатись до процесів, подій, семафорів, м’ютексів , об’єктів ядра і т.п., напряму до об’єкту звертатись не можна. Така змінна має назву дескриптора потоку.

Створення потоку відбувається за допомогою функції СreateThread(LPSECURITY\_ATTRIBUTES ThreadAttributes, DWORD StackSize, LPTHREAD\_START\_ROUTINE StartAddress, LPVOID Parameter, DWORD CreationFlags, LPDWORD ThreadID ). Функція повертає унікальний ідентифікатор потоку, приймає шість параметрів:

* LPSECURITY\_ATTRIBUTES ThreadAttributes відповідає за атрибути безпеки для створюваного потоку, оскільки кожен об’єкт ядра має свої атрибути безпеки , якщо передається NULL, то значення атрибуту безпеки встановлюється за умовчанням.
* DWORD StackSize дозволяє користувачу визначити розмір стеку , значення 0 дозволяє встановити розмір стек за замовчання(1 мегабайт).
* LPTHREAD\_START\_ROUTINE StartAddress – ім’я потокової функції ,тіло якої буде виконуватись в паралельному потоці, потік розпочне роботу з виконання даної функції, яка має бути з глобальною видимістю (об’явлена як DWORD WINAPI).
* LPVOID Parameter- аргумент функції потоку, вказівник на тип VOID.
* DWORD CreationFlags, дозволяє визначити режим старту потоку(може бути відкладений за допомогою передачі параметра Create\_Suspended), при передачу 0-за замовчуванням в системі робота потоку починається одразу після його створення.
* LPDWORD ThreadID – параметр унікальності, який забезпечує виконання кожним потоком окремого завдання. Може використовуватись повторно поки даний потік існує.

Якщо виконання функції CreateThread() не закінчилось створенням потоку, буде повернуто FALSE, причина помилки може бути визначена викликом GetLastError().Після завершення роботи потоку необхідно звільнити ресурси операційної системи викликом функції CloseHandle(HANDLE hObject), оскільки потокові HANDLE залишають зайнятою зарезервовану пам'ять і її треба звільнити до завершення роботи програми .

Значення пріоритету потоку можна отримати за допомогою виклику функції GetThreadPriority(), встановлення пріоритету для потоку відбувається шляхом виклику функції SetThreadPriority().

Організація взаємодії процесів реалізована через створення та використання семафорів, м’ютексів, критичних секцій ,таймерів, подій.

Для створення взаємодії можна використовувати HANDLE потоку, оскільки будь-який HANDLE може бути в одному з двох станів: сигналізуючому та несигналізуючому. HANDLE потоку може знаходитись в стані сигналізування   
, якщо він завершив роботу, та несигналізування в противному випадку.

В потоках, семафорах, м’ютексах , подіях широко використовується функція

WaitForSingleObject() [7]- вона виконує блокування всіх інших потоків, поки потік не закінчить виконання роботи. В якості параметрів вона приймає ідентифікатор об’єкта ,часовий ліміт очікування(якщо час очікування перевищено, то код стає доступним для виконання іншими потоками), параметр раннього виконання функції. Також існує функція WaitForMultipleObjects()-в якій можна очікувати всі передані потоки(fWaitAll==TRUE), або хоч одного(fWaitAll==FALSE).

М’ютекси використовуються для вирішення задач взаємовиключення та синхронізації . М’ютекс в момент часу може бути доступний лише одному потоку для контрою доступу до спільного ресурсу. Для створення м’ютексу реалізована функція CreateMutex(), яка повертає HANDLE м’ютексу, для блокування м’ютексу викликається функція WaitForSingleObject(), якщо м’ютекс не заблокований то його HANDLE знаходиться в стані сигналізації, при виконанні функції м’ютекс захоплюється одним з потоків, його HANDLE переходить в несигналізуючий стан.

Для звільнення м’ютексу використовуються функція ReleaseMutex().

При використанні критичних ділянок використовується механізм критичних секцій Win32. Для визначення критичної секції необхідно об’явити змінну типу CRITICAL\_SECTION , для створення та видалення використати оператори InitializeCriticalSection(&cs), DeleteCriticalSection(&cs). Вхід в критичну ділянку описується оператором EnterCriticalSection(&*cs*), який блокує роботу всіх інших потоків, якщо якийсь потік вже знаходиться в критичній секції чи дозволяє процесу зайти в критичну секцію , якщо вона вільна. Для виходу з критичної секції викликається функція LeaveCriticalSection(&cs).

Також важливим інструментов для вирішення задач взаємодії є семафор. Семафори створюється функцією HANDLE CreateSemaphore(LPSECURITY\_ATTRIBUTES lpEventAttributes, LONG lSemInitial, LONG lSemMax , LPCSTR lpSemName)- в якості параметрів вона приймає атрибут безпеки, початкове значення лічильника, кінцеве значення лічильника, ім’я семафору. Значення lSemMax має бути , а початкове значення лічильника має бути і не може виходити за границі діапазону. Для операції зменшення лічильника семафору використовується функція WaitForSingleObject, яка зменшує лічильник на 1, для збільшення лічильника семафору використовується функція BOOL ReleaseSemaphore().



Бібліотека WinAPI є гнучким та функціональним засобом створення паралельних комп’ютерних програм на ОС Windows. WinAPI має всі функції , необхідні для створення ,конфігурації, організації взаємодії паралельних потоків.

**1.5 Процеси в бібліотеці POSIX Threads**

POSIX threads- бібліотека , реалізована згідно стандарту POSIX 1003.b, який визначає API(інтерфейс прикладного програмування ). POSIX дозволяє створювати паралельні процеси та виконувати конфігурацію їх дій на ОС Linux, оскільки POSIX є розширенням механізму потоків , створеного в цій ОС.

Pthreads представляє собою прикладний програмний інтерфейс для виконання більшості дій, необхідних для функціонування паралельних потоків. Бібліотека має функції для створення, переривання ,блокування , взаємодії потоків.

Взаємодія організовується за допомогою м’ютексів, семафорів, бар’єрів.

Розглянемо основні функції для створення та конфігурації процесів в POSIX[8]:

* pthread\_create (thread, attr , start\_routine, arg)- функція створення нового потоку. Після створення потік виконується паралельно головному. В якості параметрів передається унікальний ідентифікатор потоку, об’єкт атрибутів(якщо передано Null-то атрибути встановлюються за замовчуванням), потокова функція та аргумент потокової функції.
* pthread\_attr\_init(),pthread\_attr\_destroy()-функції,які використовується для створення чи видалення об’єкту з атрибутами потоку.
* pthread\_exit()-функція виходу з потоку після його завершення, може викликатись автоматично після нормального завершення дій потоку, після безпосереднього виклику функції процесом, незалежно завершив він свою роботу чи ні, при використанні функції pthread\_cancel()-виконання потоку відміняється через виклик даної функції.
* pthread\_mutex\_init(mutex,attr)-створення м’ютексу з певними атрибутами .
* pthread\_mutex\_destroy() –використовується для звільнення м’ютексу.
* pthread\_mutex\_lock()-функція для блокування тих потоків, які звертаються до м’ютексу переданого в якості параметру.
* pthread\_mutex\_unlock() –функція , яка виконає розблокування м’ютексу при виклику в потоці, який володіє м’ютексом .
* pthread\_self()-повертає ідентифікатор потоку, в якому була викликана.

В POSIX будь-який м’ютекс буває трьох типів normal, recursive, error check[9].

Нормальний м’ютекс можна захопити лише один раз в одному потоці, потворне захоплення спричинить появу помилки. Рекурсивний м’ютекс можна захопити n разів, але його він має бути звільнений також n разів, цей тип м’ютексу працює доволі повільно. М’ютекс з перевіркою помилок має декілька додаткових кодів помилок. Pthread\_mutex\_lock() виконує миттєве захоплення м’ютексу, якщо він вільний , якщо ні –то функція чекає його звільнення викликом функції pthread\_mutex\_unlock() . Код між викликами цих функцій є критичною ділянкою в POSIX threads.

POSIX – інструмент для програмування паралельних комп’ютерних систем на базі операційної системи Linux та інших Unix-подібних операційних систем.

**1.6. Висновки до розділу 1**

1. Виконано аналіз засобів створення, взаємодії процесів в бібліотеці OpenMP. До переваг можна віднести можливість швидкого розпаралелювання циклів, можливість швидкого створення паралельної програми з послідовної шляхом додання директив.

До недоліків можна віднести організацію взаємодії процесів через спільні змінні , а не через передачу повідомлень.

2. Виконано огляд бібліотечних функцій та засобів бібліотеки MPI. Перевагами MPI є можливість вирішення проблеми переносу паралельних програм між різними комп’ютерними системами, створення легкомасштабуємих програм, підвищення ефективності паралельних обчислень.

До недоліків можна віднести відносну складність в освоєнні інтерфейсу обміну повідомленнями, низькорівневість.

3.Проведено огляд засобів бібліотеки паралельного програмування PVM. Переваги: PVM є простішим за MPI, більш гнучким, ніж OpenMP, дозволяє роботу в гетерогенних середовищах. Недоліки: малопоширеність, відносно важкий API, необхідність складної конфігурації процесів при написанні програми.

4.Виконано аналіз інструментів створення та взаємодії процесів в бібліотеці POSIX Threads. Переваги даної бібліотеки: стандарт POSIX підтримує майже кожна Unix-подібна система, підтримка засобів синхронізації. До недоліків відносяться: низькорівневість, складність в освоюванні , деякі реалізації недостатньо оптимізовані.

5.Проаналізовано процеси, їх характеристики та взаємодія в бібліотеці WinAPI. Дана бібліотека надає користувачу широкі можливості для паралельного програмування. Перевагами є: велика кількість інструментів для вирішення задач взаємодії між процесами, швидкодія. Недоліки: низькорівневість.

# РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС СП

**МЕЖСТРОЧНЫЙ ИНТЕРВАЛ ВЕЗДЕ 1,5!!**

Математична задача:



Структура ПКС СП зображена на рис. 2.1. Від і вивід даних відбувається в процессах 1,5,8.

СП

## 

8

5

1

## … …

## 

А, B, E MO, MK, MT



Рис. 2.1 Структура ПКС з СП

## 2.1. Розробка паралельного математичного алгоритму

Згідно з технічним завданням необхідно розробити паралельний алгоритм. Кількість процесорів P=8. Його можна розділити на наступні етапи:

1. *СР: B.*



1. *; CР: T, MK,* .



Пояснення до використовуваних констант:

*  – розмірність векторів і матриць;
*  – кількість ядер;
* .



Оцінка прискорення і ефективність розробленого алгоритму, спираючись на теорему Мунро-Петерсона:



де – кількість бінарних операцій;



– час розв’язання задачі на Р вузлах.



1. Операція множення двох матриць розмірністю потребує операцій множення і операцій додавання для формування одного елемента матриці MA. Отже, , .



1. Операція множення вектора на матрицю потребує N-1 операцій додавання. .



1. Операція множення вектора на число потребує операцій множення , отже



1. Операція додавання виконується за один такт .



Отже, сумарний час виконання операції:



Якщо , то .



**2.2 Розробка алгоритмів процесів**

ЗадачаТ1: ТСКД

1. Введення даних B,E.
2. Cигнал про введення даних процесам Т2-Т8.



1. Очікування введення в Т5,Т8.



1. Копіювання B1=B; КД

5)Обчислення



6)Сигнал про обчислення процесам Т2-Т8.



7)Очікування обчислення від Т2-Т8.



8)Копіювання *T1=Т, MK1=МК,* . КД



9)Обчислення



10)Очікування обчислення від Т2-Т8.



11).Виведення А.

Задача Т5:

1)Введення даних MO,.



2)Cигнал про введення даних процесам Т1-Т4,Т6-Т8.

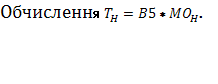
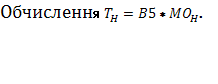


3)Очікування введення в Т1,Т8.



4)Копіювання B5=B; КД

5)



6) Сигнал про обчислення процесам Т1-Т4,Т6-Т8.



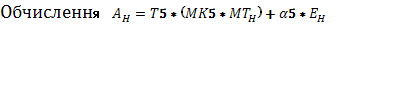
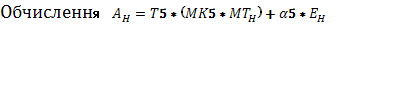
7) Очікування обчислення від Т1-Т4,Т6-Т8.



8)Копіювання *T5=Т, MK5=МК,* . КД



9).



10)Сигнал про обчислення Т1.



Задача Т8:

1)Введення даних MК.



2)Cигнал про введення даних процесам Т1-Т7.

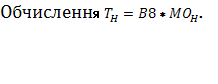
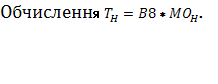


3)Очікування введення в Т1,Т5.



4)Копіювання B8=B; КД

5)



6) Сигнал про обчислення процесам Т1-Т7.



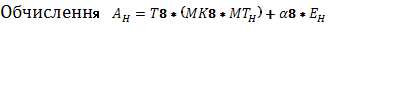
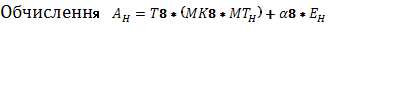
7) Очікування обчислення від Т1-Т7.



8)Копіювання *T8=Т, MK8=МК,* . КД



9).



10)Сигнал про обчислення Т1.



Задача i=()



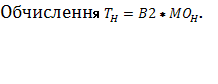
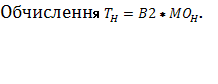
Задачі Т3,Т4,Т6,Т7 мають аналогічний алгоритм з використанням заміни індексу на відвідний номер задачі, індекс

1)Очікування введення в Т1,Т5,Т8.



2)Копіювання B2=B; КД

3)



4) Сигнал про обчислення процесам Т1,Т3-Т8.



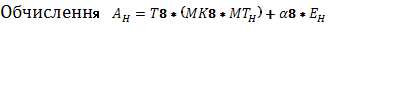
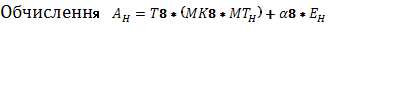
5) Очікування обчислення від Т1-Т4,Т6-Т8.



6)Копіювання *T8=Т, MK8=МК,* . КД



7).



8)Сигнал про обчислення Т1.



**2.3 Розробка схеми взаємодії**

Для коректних обчислень необхідне використання двох захищених модулів. Перший (DataMonitore) відповідає за вирішення задачі взаємного виключення(доступ до спільних ресурсів), другий(SyncMonitore)- за синхронізацію задач.

В DataMonitore використовується захищені функції та захищені процедури.

Оскільки захищені функції дозволяють лише зчитувати дані, вони ідеально підходять для копіюванні даних при доступі до СР. Для копіювання необхідно створити 4 захищені функції : CopyB(копіювання вектора B), CopyT(копіювання вектора Т), CopyMK(копіювання матриці MK), CopyAlpha(коп).



Зміни спільних ресурсів можливі лише при використанні захищених процедур ,які дозволяють проводити зчитування та запис даних . В програмі використовуються захищені процедур запису в монітор WriteB(запис вектора B), WriteMK(запис матриці MK),WriteAlpha(),WriteT(запис T). Схема роботи описана на рис 2.2.



Cхема роботи SyncMonitore наведена на рис 2.3.В моніторі відбувається синхронізація потоків за допомогою захищених процедур та захищених входів. Захищені процедури відповідають за сигнали про певну подію, захищені входи дозволяють очікування певного сигналу.

Захищені процедури: InputSignal(сигнал про введення в потоках Т1,Т5,Т8), CalculatingT(сигнал про обчислення ), CalculatingA(Cигнал про обчислення ).



Захищені входи: InputtingWait(очікування вводу), WaitCalcT (очікування обчислення ), WaitCalcA(очікування обчислення ).



**2.4 Розробка програми ПРГ1**

Програма розроблена на мові Ада з використанням захищених модулів. Головною є процедура Main в якій описані константи: N-розмірність матриць ,H-частина розміром N/P,P-кількість процесів .Також дана процедура містить специфікації та реалізації двох захищених модулів DataMonitore та SyncMonitore , які використовуються для вирішення задач взаємного виключення та задачі синхронізації. Дії потоків реалізовані в процедурі Tasks, яка містить реалізації задач Т1-Т8.

Процедури вводу ,копіювання та виводу знаходяться в головній процедурі.

Лістинг програми знаходиться в додатку А . Алгоритми взаємодії потоків знаходяться в додатку Б.

**2.5 Тестування програми ПРГ1**

Метою проведення тестування є оцінка коефіцієнтів прискорення і коефіцієнтів ефективності для розроблених програм при їх виконанні на реальній паралельній обчислювальній системі. Для обчислень використовуються різні значення розмірностей величин , якими оперує програма (N = 1200, 1500, 1800, 2100) і різною кількістю працюючих ядер ( P = 1, 2, 3, 4).

DataMonitore

B

CopyB



WriteB



T

CopyT



WriteT



MK

CopyMK



WriteMK



CopyAlpha



WriteAlpha



Рис 2.2 Структурна схема монітора для вирішення задачі взаємного виключення

SyncMonitore

F1=3

F1=F1+1

F2=F2+1

F3=F3+1

InputtingWait

InputtingSignal



F2=8 F2=8



WaitCalcT

CalculatingT



F3=7



WaitCalcA

CalculatingA



Рис 2.3 Структурна схема монітора для вирішення задачі синхронізації

Для оцінки часу виконання програми використовуються процедура Clock з пакету Calendar. Час виконання отримується за рахунок обчислення різниці кінцевого та часу запуску програми.

Для оцінки ефективності програми використовуються коефіцієнти прискорення та ефективності.

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



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



Результати тестування і проведених досліджень ефективності розробленої програми наведено в табл. 2.1-2.3.

Таблиця 2.1. Час виконання програми ПРГ1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | T1 | T2 | T3 | T4 |
| 1200 | 72 | 55 | 42 | 39 |
| 1600 | 114 | 101 | 67 | 62 |
| 1800 | 219 | 158 | 114 | 106 |
| 2000 | 254 | 165 | 157 | 148 |

Використовуючи дані з таблиці 2.1 розраховано коефіцієнти прискорення в ПРГ1. РЕЗУЛЬТАТЫ ПЛОХИЕ!!! НЕ ПОЙДЕТ!!!!

Таблиця 2.2. Коефіцієнти прискорення для програми ПРГ1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | | | |
| 1 | 2 | 3 | 4 |
| 1200 | 1 | 1,3 | 1,7 | 1,8 ???????? |
| 1600 | 1 | 1,12 | 1,7 | 1,83 |
| 1800 | 1 | 1,38 | 1,92 | 2,06 |
| 2100 | 1 | 1,53 | 1,61 | 1,71 |

За допомогою коефіцієнтів прискорення обчислено коефіцієнти ефективності в таблиці 2.3.

Таблиця 2.3. Коефіцієнти ефективності для програми ПРГ1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | P | | | |
| 1 | 2 | 3 | 4 |
| 1200 | 100 | 65 | 56 | 45 |
| 1600 | 100 | 56 | 56 | 51,5 |
| 1800 | 100 | 69 | 64 | 49,5 |
| 2100 | 100 | 76,5 | 87 | 43 |

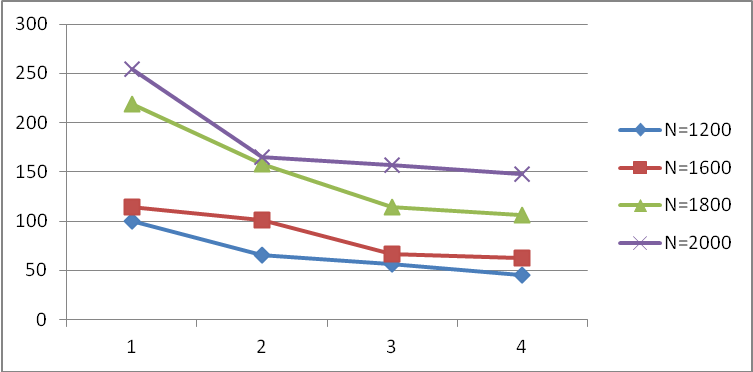


Рис. 2.4. Графік зміни часу виконання обчислень програми ПРГ1 в залежності від кількості ядер

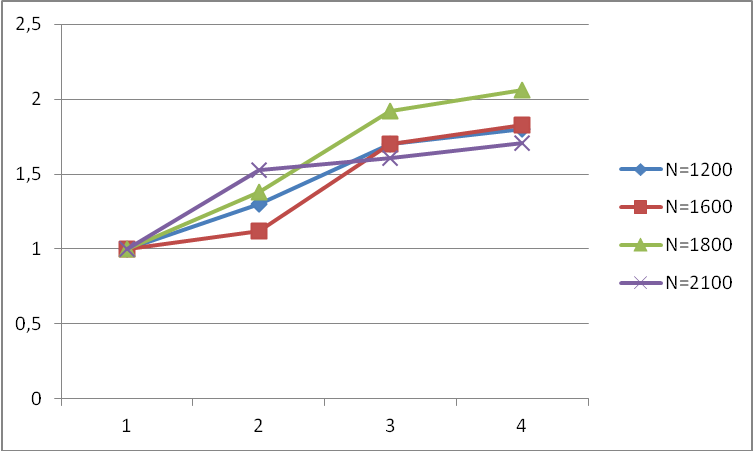


Рис. 2.5. Графік зміни коефіцієнту прискорення програми ПРГ1 в залежності від кількості ядер

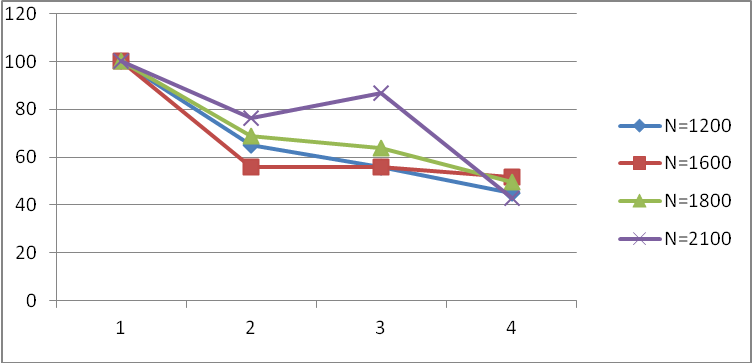


Рис. 2.6. Графік зміни коефіцієнту ефективності програми ПРГ1 в залежності від кількості ядер

**2.6 Висновки до розділу 2.**

* Максимальне значення  = 2,06 забезпечує ПКС з Р = 4 та N=1800.
* Мінімальне значення  = 1,12 забезпечує ПКС з Р = 2 та N = 1500.
* По графіку з рис. 2.4 видно, що коефіцієнт прискорення збільшується прапорційно до кількості використовуваних ядер.
* Коефіцієнти ефективності коливаються в межах від 81,9-100%, це свідчить про високу ефективність розробленого паралельного алгоритму.
* Максимальне значення = 87 для ПКС з Р = 3 та N = 2100;
* Мінімальне значення = 43% для ПКС з Р = 4 та N = 2100.

Оцінивши значення  та  можна зробити висновок про доцільність використання ресурсів ПКС для роз`в’язання задач такого типу.

# РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП

В розділі розглянуто розробку та дослідження ефективності роботи програми ПРГ2 для системи з спільною пам’яттю. Програма виконується мовою програмування Java з використанням бібліотеки паралельного програмування MPI.

Математична задача . Для взаємодії процесів використовується механізм передачі повідомлень.

Структура ПКС ЛП зображена на рис. 3.1. Від і вивід даних відбувається в процессах 1,5,8.

4

3

2

1

MK,MT

MO,



A,B,E

5

6

7

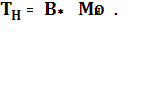
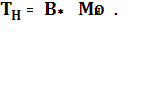
8

Рис. 3.1 Структура ПКС із локальною пам’яттю

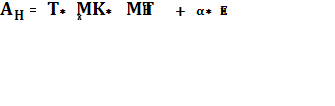
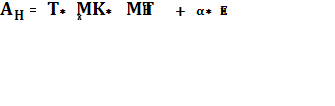
**3.1 Розробка паралельного математичного алгоритму**

Згідно з технічним завданням необхідно розробити паралельний алгоритм. Кількість процесорів P=8. Етапи обчислень:

1. СР: B.



1. ; CР: T, MK, .



Використані константи:

1) – розмірність векторів і матриць;

2) – кількість ядер;

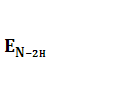
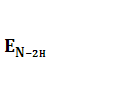
3);



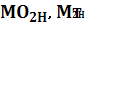
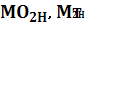
**3.2 Розробка алгоритмів процесів**

**Задача Т1:**

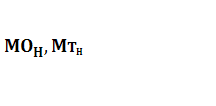
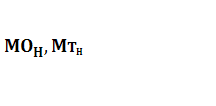
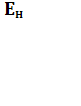
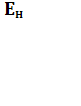
1. Введення B,E.
2. Передати B, в Т2



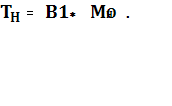
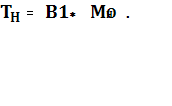
1. Прийняти MK,, від Т2



1. Передати B,,MK,, в Т5.



1. Обчислення 1:



1. Прийняти від Т2.



1. Прийняти від Т5.



1. Передати в Т5.



1. Передати в Т2.



1. Обчислення 2:



10).Прийняти від Т2.



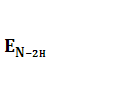
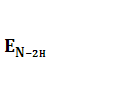
11).Прийняти від Т5.



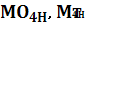
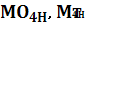
12).Вивести А.

**Задача Т2:**

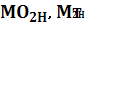
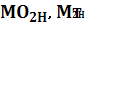
1. Прийняти B, від Т1



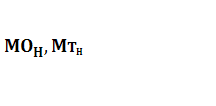
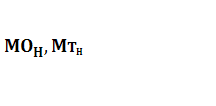
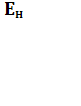
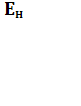
1. Прийняти MK,, від Т3



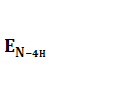
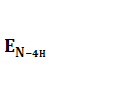
1. Передати MK,, в Т1.



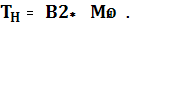
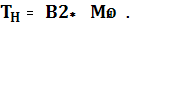
1. Передати B,,MK,, в Т6.



1. Передати B, в Т3.



1. Обчислення 1:



1. Прийняти від Т6.



1. Прийняти від Т1.



1. Прийняти від Т3.



10)Передати в Т1.



11)Передати в Т6.



12)Передати в Т3.



13) Обчислення 2:



14)Прийняти

15)Прийняти від Т6.

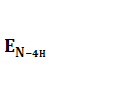
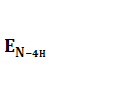


16)Передати в Т1.

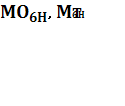
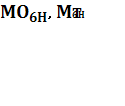


**Задача Т3:**

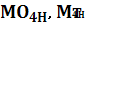
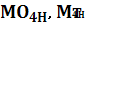
1. Прийняти B, від Т2.



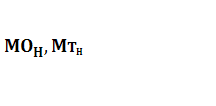
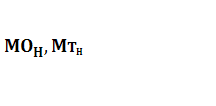
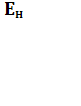
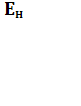
1. Прийняти MK,, від Т4



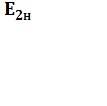
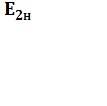
1. Передати MK,, в Т2.



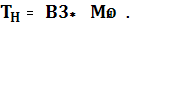
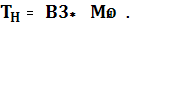
1. Передати B,,MK,, в Т7.



1. Передати B, в Т4.



1. Обчислення 1:



1. Прийняти від Т4.



1. Прийняти від Т2.



1. Прийняти

10)Передати в Т2.



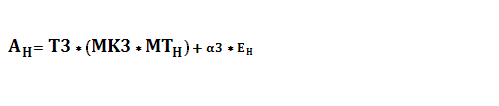
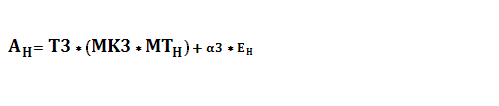
11)Передати .



12)Передати в Т7.



13)Обчислення 2:



14)Прийняти від Т4.



15)Прийняти Т7.

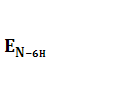
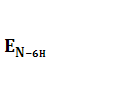
16)Відправити в Т2.



**Задача Т4:**

1)Введення MK,MT.

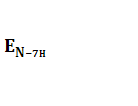
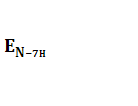
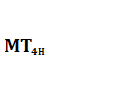
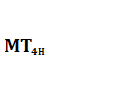
2) Прийняти B, від Т3.



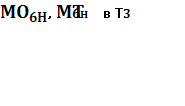
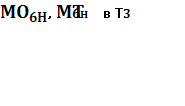
3)Прийняти , від Т8.



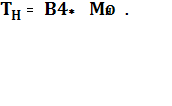
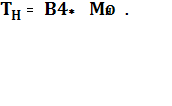
4) Передати MK, , B,в Т8.



5) Передати MK,,.



6) Обчислення 1:



7)Прийняти від Т3.



8)Прийняти від Т8.



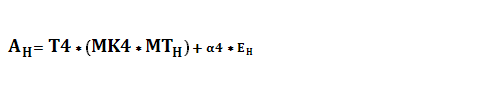
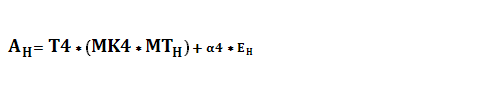
9)Передати в Т3.



10)Передати в Т8.



11)Обчислення 2:



12)Прийняти від Т8.

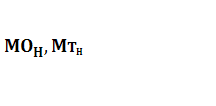
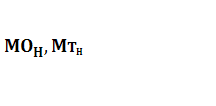
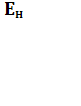
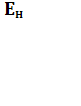


13)Передати

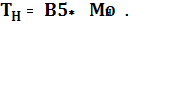
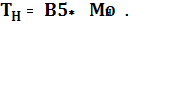


**Задача Т5:**

1)Прийняти B,,MK,, від Т1.



2)Обчислення 1:



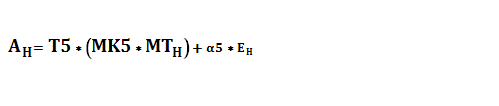
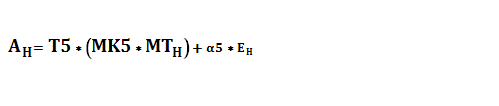
3)Передати в Т1.



4)Прийняти від Т1.



5)Обчислення 2:

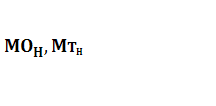
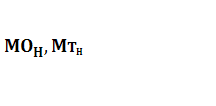
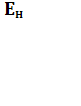
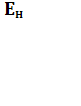


6)Передати в Т1.

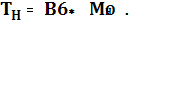
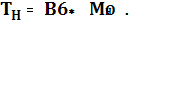


**Задача Т6:**

1)Прийняти B,,MK,, від Т2.



2)Обчислення 1:



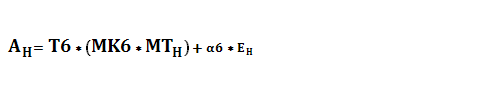
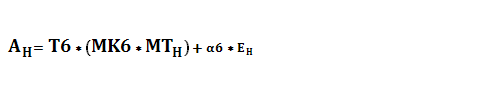
3)Передати в Т2.



4)Прийняти від Т2.



5)Обчислення 2:

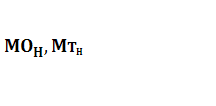
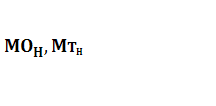
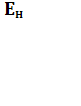
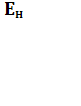


6)Передати в Т2.

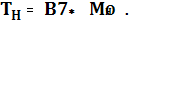
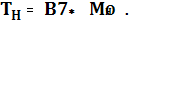


**Задача Т7:**

1)Прийняти B,,MK,, від Т3.



2)Обчислення 1:



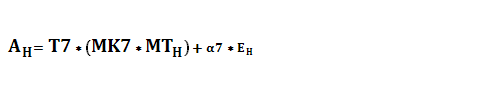
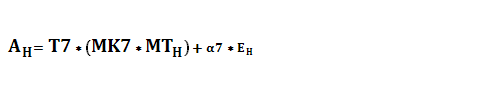
3)Передати в Т3.



4)Прийняти від Т3.



5)Обчислення 2:



6)Передати в Т3.



**Задача Т8:**

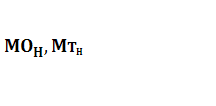
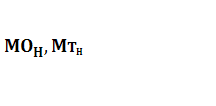
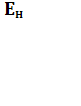
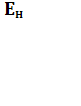
1)Введення MO,.



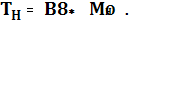
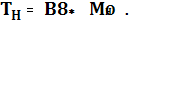
2)Передати ,



3)Прийняти B,,MK,, від Т4.



4)Обчислення 1:



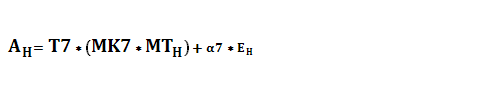
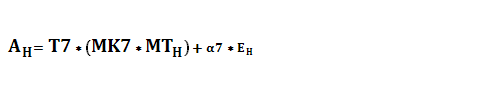
5)Передати в Т.



6)Прийняти від Т3.



7)Обчислення 2:



8)Передати в Т3.



## 3.3. Розробка схеми взаємодії процесів

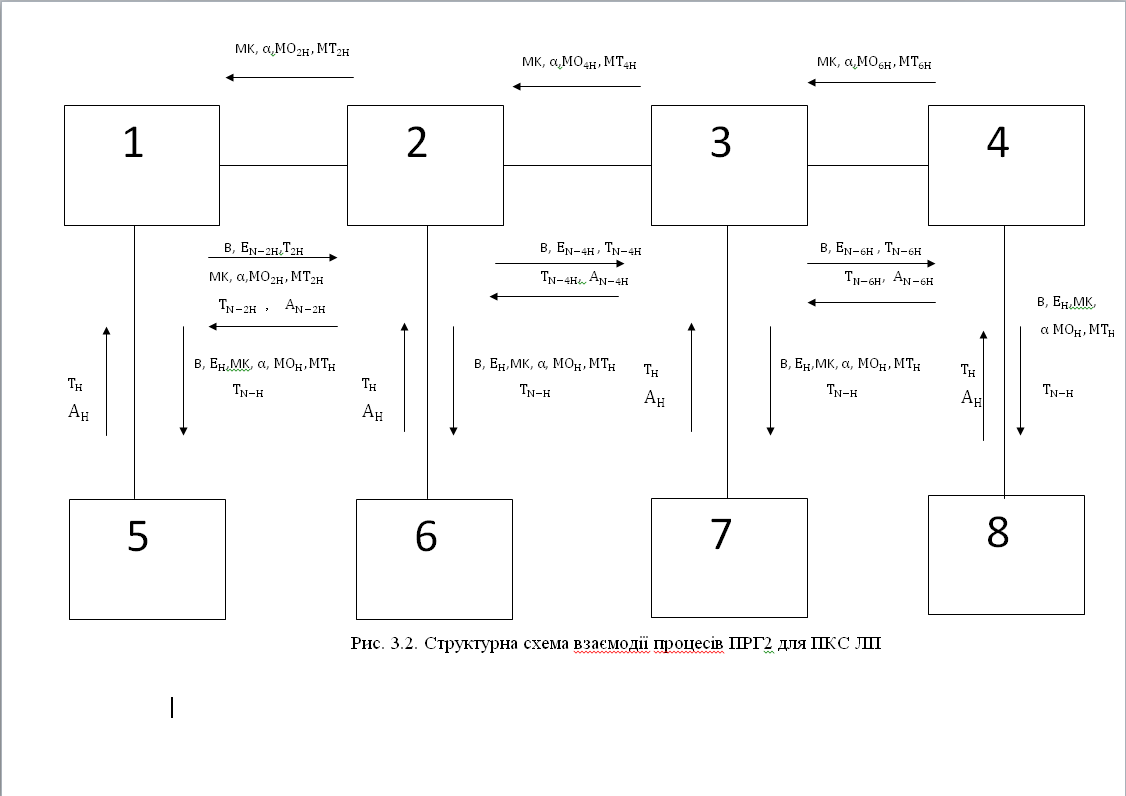
На основі вищенаведених алгоритмів кожного процесу побудовано схему взаємодії процесів(рис 3.2).

Введення даних виконується в задачах 1,4,8 . Задачі 2 та 3 мають відсилати та приймати найбільшу кількість повідомлень, оскільки вони пов’язані з трьома задачами сусідами. Для організації взаємодії використовується базові функції MPI:

1)MPI\_send – функція викликає передачу даних в вигляді повідомлення.

2)MPI\_recv– функція прийому повідомлення

?



**3.4 Розробка програми ПРГ 2**

В програмі використовуються дві базові функції для обміну повідомленнями MPI\_send – яка блокує процес ,який відправляє повідомлення поки повідомлення не буде отримане та MPI\_recv– функція прийому повідомлення . В процесах 1,4,8 відбувається введення даних ,дані процеси розсилають всім іншим та один одному введені дані.

Обчислення відбувається в два етапи, виведення результату є доцільним для N<=24.

**3.5 Тестування ПРГ2**

# ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ПО РОБОТІ

1. Проведено огляд бібліотек параллельного програмування MPI, POSIX , OpenMP, PVM, Win32, показано засоби створення процесів та методи взаємодії між ними. Паралельні процеси можуть бути створені за допомогою потокових функцій(Win32,POSIX) та щляхом копіювання послідовної ділянки програми(MPI, OpenMP,PVM). Основними засобами взаємодії паралельних процесів є м’ютекси, монітори, семафори, критичні секції, механізм обміну повідомленнями. Кожна з бібліотек має свою реалізацію цих засобів. Також наведені переваги та недоліки використання кожної з бібліотек.

2. Виконано розробку двух паралельних програм для систем з СП та ЛП. Програми виконують обчислення за встановленою математичною задачею . Програма для системи з СП з використанням механізму захищених модулів показує кращі показники ефективності , ніж програма написана для системи з ЛП, оскільки механізм обміну повідомлення впроваджує пересилку великих блоків пам’яті , що є дуже затратним процесом.

3. Програма для системи з СП займає в пам’яті набагато менше місця, ніж програма з СП, оскільки копіювання всіх спільних ресурсів не відбувається для кожного процесу.

4. Програми з СП є більш простими для розуміння та написання , що дозволяє якісніше та швидше відлагодждувати програму та підтримувати її працездатність.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. OpenMP[Електронний ресурс] .— Режим доступу:

http://openmp.org/wp/openmp-specifications/— дата

звернення: 22.02.2015.— Назва з екрану.

2. Параллельное програмирование с OpenMP[Електронний ресурс] .— Режим доступу: http://www.intuit.ru/studies/courses/1112/232/lecture/6025?page=1— дата

звернення: 22.02.2015.— Назва з екрану.

3. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення: Навч.

посібник. – К.: Корнійчук, 2014. – 284 с.

4. Технологии параллельного программирования.MPI [Електронний ресурс] .— Режим доступу:http://parallel.ru/vvv/mpi.html— дата

звернення: 22.02.2015.— Назва з екрану.

5. Использование PVM. Введение в программирование [Електронний ресурс] .—Режим доступу:

http://chaos.ssu.runnet.ru/dynamics/books/pvm/using\_PVM.htm— дата звернення:

22.02.2015.— Назва з екрану.

6. PVM Manual[Електронний ресурс] .— Режим доступу:

http://www.csm.ornl.gov/pvm/man/— дата звернення: 22.02.2015.— Назва з екрану.

7. ОС и аспекты параллельного программирования[Електронний ресурс] .— Режим доступу: http://www.intuit.ru/studies/courses/4447/983/lecture/14923— дата звернення: 22.02.2014.— Назва з екрану.

8. POSIX Threads Programming [Електронний ресурс] .— Режим доступу:

https://computing.llnl.gov/tutorials/pthreads/ — дата звернення:

22.02.2015.— Назва з екрану.

9. POSIX threads(pthreads)libraries [Електронний ресурс] .— Режим доступу:

http://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html дата звернення:

22.02.2015.— Назва з екрану.

**ДОДАТКИ**

**Додаток А**

**Лістинг програми ПРГ1**

GNAT GPL 2013 (20130314)

Copyright 1992-2013, Free Software Foundation, Inc.

Compiling: main.adb (source file time stamp: 2015-03-22 09:57:28)

--Serhin Dmytro

--Ada.Rozdil 2(zasch.modul)

--IO-22

--A=(B\*MO)(MK\*MT)+alpha\*E

1. with Ada.Text\_IO, Ada.Integer\_Text\_IO, Ada.Calendar;

2. use Ada.Text\_IO, Ada.Integer\_Text\_IO, Ada.Calendar;

3.

4.

5. procedure Main is

6.

7. N:integer:=8;

8. P:integer:=8;

9. H:integer:=N/P;

10.

11. StartTime, FinishTime: Time;

12. Delta: Duration;

13.

14.

15. type Vector is array(1..N)of integer;

16. type Matrix is array(1..N)of Vector;

17.

18. A,B,E,T:Vector;

19. MO,MK,MT:Matrix;

20. fill:integer:=1;

21. alpha:integer:=fill;

22.

23. procedure MatrixInput(M: out Matrix) is

24. begin

25. for i in 1..N loop

26. for j in 1..N loop

27. M(i)(j) := fill;

28. end loop;

29. end loop;

30. end MatrixInput;

31.

32. procedure Copy\_Matrix(InputMatrix:in Matrix;OutputMatrix:out Matrix)is

33. begin

34. for i in 1..N loop

35. for j in 1..N loop

36. OutputMatrix(i)(j):=InputMatrix(i)(j);

37. end loop;

38. end loop;

39. end;

40.

41. procedure Copy\_Vector(InputV:in Vector;OutV:out Vector)is

42. begin

43. for i in 1..N loop

44. OutV(i):=InputV(i);

45. end loop;

46. end;

47.

48. procedure VectorInput(V: out Vector) is

49. begin

50. for i in 1..N loop

51. V(i):=fill;

52. end loop;

53. end VectorInput;

54.

55. procedure VectorOutput(V:in Vector)is

56. begin

57. for i in 1..N loop

58. Put(V(i));

59. Put(" ");

60. end loop;

61. end VectorOutput;

62.

63. protected DataMonitore is

64. function copyB return Vector;

65. function copyT return Vector;

66. function copyMK return Matrix;

67. function copyAlpha return integer;

68. procedure writeB(V:in Vector);

69. procedure writeT(V:in Vector);

70. procedure writeMK(M:in Matrix);

71. procedure writeAlpha(al:in integer);

72. private

73. B:Vector;

74. T:Vector;

75. MK:Matrix;

76. alpha:integer;

77. end DataMonitore;

78.

79. protected SyncMonitore is

80. procedure InputSignal;

81. procedure CalculatingT;

82. procedure CalculatingA;

83. entry InputtingWait;

84. entry WaitCalcT;

85. entry WaitCalcA;

86. private

87. f1:integer:=0;

88. f2:integer:=0;

89. f3:integer:=0;

90. end SyncMonitore;

91.

92. protected body DataMonitore is

93. procedure writeB(V:in Vector) is

94. begin

95. B:=V;

96. end writeB;

97.

98. procedure writeT(V:in Vector)is

99. begin

100. T:=V;

101. end writeT;

102.

103. procedure writeMK(M:in Matrix)is

104. begin

105. MK:=M;

106. end writeMK;

107.

108. procedure writeAlpha(al:in integer)is

109. begin

110. alpha:=al;

111. end writeAlpha;

112.

113. function copyB return Vector is

114. begin

115. return B;

116. end;

117.

118. function copyT return Vector is

119. begin

120. return T;

121. end;

122.

123. function copyMK return Matrix is

124. begin

125. return MK;

126. end;

127.

128. function copyAlpha return integer is

129. begin

130. return alpha;

131. end;

132. end DataMonitore;

133.

134. protected body SyncMonitore is

135. procedure InputSignal is

136. begin

137. f1:=f1+1;

138. end;

139.

140. procedure CalculatingT is

141. begin

142. f2:=f2+1;

143. end;

144.

145. procedure CalculatingA is

146. begin

147. f3:=f3+1;

148. end;

149.

150. entry InputtingWait

151. when f1=3 is

152. begin

153. null;

154. end;

155.

156. entry WaitCalcT

157. when f2=8 is

158. begin

159. null;

160. end;

161.

162. entry WaitCalcA

163. when f3=7 is

164. begin

165. null;

166. end;

167.

168. end SyncMonitore;

169. procedure Tasks is

170. task t1 is

171. end t1;

172. task body T1 is

173. B1,T1:Vector;

174. alpha1:integer;

175. MK1:Matrix;

176. temp,temp1:integer:=0;

177. begin

178. Put\_line("Task t1 is started");

179. VectorInput(B);

180. VectorInput(E);

181.

182. DataMonitore.writeB(B);

183.

184. SyncMonitore.InputSignal;

185. SyncMonitore.InputtingWait;

186. B1:=DataMonitore.copyB;

187. for i in 1..H loop

188. temp:=0;

189. for j in 1..N loop

190. temp:=temp+B1(j)\*MO(i)(j);

191. end loop;

192. T(i):=temp;

193.

194. end loop;

195.

196. SyncMonitore.CalculatingT;

197. SyncMonitore.WaitCalcT;

198.

199. T1:=DataMonitore.copyT;

200. MK1:=DataMonitore.copyMK;

201. alpha1:=DataMonitore.copyAlpha;

202.

203. for i in 1..H loop

204. for j in 1..N loop

205. temp1:=0;

206. for k in 1..N loop

207. temp1:=temp1+T1(j)\*(MT(i)(k)\*MK1(k)(j))+alpha1\*E(i);

208. end loop;

209. end loop;

210. A(i):=temp1;

211. end loop;

212. SyncMonitore.WaitCalcA;

213. if(N<16) then

214. VectorOutput(A);

215. end if;

216. Put\_Line("T1 is finished");

217. FinishTime := Clock;

218. DiffTime := FinishTime - StartTime;

219. Put("Time = ");

220. Put(Integer(Delta), 1);

221. Put\_Line("");

222. end t1;

223. task t2 is

224. end t2;

225. task body t2 is

226. B2,T2:Vector;

227. alpha2:integer;

228. MK2:Matrix;

229. temp,temp1:integer:=0;

230. begin

231. Put\_line("T2 is started");

232.

233. SyncMonitore.InputtingWait;

234. B2:=DataMonitore.copyB;

235.

236. for i in H+1..2\*H loop

237. temp:=0;

238. for j in 1..N loop

239. temp:=temp+B2(j)\*MO(i)(j);

240. end loop;

241. T(i):=temp;

242. end loop;

243.

244. SyncMonitore.CalculatingT;

245. SyncMonitore.WaitCalcT;

246.

247. T2:=DataMonitore.copyT;

248. alpha2:=DataMonitore.copyAlpha;

249. MK2:=DataMonitore.copyMK;

250.

251. for i in H+1..H\*2 loop

252. for j in 1..N loop

253. temp1:=0;

254. for k in 1..N loop

255. temp1:=temp1+T2(j)\*(MT(i)(k)\*MK2(k)(j))+alpha2\*E(i);

256. end loop;

257. end loop;

258. A(i):=temp1;

259. end loop;

260.

261. SyncMonitore.CalculatingA;

262. Put\_line("T2 is ended");

263. end t2;

264.

265. task t3 is

266. end t3;

267. task body t3 is

268. B3,T3:Vector;

269. alpha3:integer;

270. MK3:Matrix;

271. temp,temp1:integer:=0;

272. begin

273. Put\_line("T3 is started");

274.

275. SyncMonitore.InputtingWait;

276. B3:=DataMonitore.copyB;

277.

278. for i in 2\*H+1..3\*H loop

279. temp:=0;

280. for j in 1..N loop

281. temp:=temp+B3(j)\*MO(i)(j);

282. end loop;

283. T(i):=temp;

284. end loop;

285.

286. SyncMonitore.CalculatingT;

287. SyncMonitore.WaitCalcT;

288.

289. T3:=DataMonitore.copyT;

290. alpha3:=DataMonitore.copyAlpha;

291. MK3:=DataMonitore.copyMK;

292.

293. for i in 2\*H+1..H\*3 loop

294. for j in 1..N loop

295. temp1:=0;

296. for k in 1..N loop

297. temp1:=temp1+T3(j)\*(MT(i)(k)\*MK3(k)(j))+alpha3\*E(i);

298. end loop;

299.

300. end loop;

301. A(i):=temp1;

302. end loop;

303.

304. SyncMonitore.CalculatingA;

305. Put\_line("T3 is ended");

306. end t3;

307.

308. task t4 is

309. end t4;

310. task body t4 is

311. B4,T4:Vector;

312. alpha4:integer;

313. MK4:Matrix;

314. temp,temp1:integer:=0;

315. begin

316. Put\_line("T4 is started");

317.

318. SyncMonitore.InputtingWait;

319. B4:=DataMonitore.copyB;

320.

321. for i in 3\*H+1..4\*H loop

322. temp:=0;

323. for j in 1..N loop

324. temp:=temp+B4(j)\*MO(i)(j);

325. end loop;

326. T(i):=temp;

327. end loop;

328.

329. SyncMonitore.CalculatingT;

330. SyncMonitore.WaitCalcT;

331.

332. T4:=DataMonitore.copyT;

333. alpha4:=DataMonitore.copyAlpha;

334. MK4:=DataMonitore.copyMK;

335.

336. for i in 3\*H+1..H\*4 loop

337. for j in 1..N loop

338. temp1:=0;

339. for k in 1..N loop

340. temp1:=temp1+T4(j)\*(MT(i)(k)\*MK4(k)(j))+alpha4\*E(i);

341. end loop;

342. end loop;

343. A(i):=temp1;

344. end loop;

345.

346. SyncMonitore.CalculatingA;

347. Put\_line("T4 is ended");

348. end t4;

349.

350. task t5 is

351. end t5;

352. task body t5 is

353. B5,T5:Vector;

354. alpha5:integer;

355. MK5:Matrix;

356. temp,temp1:integer:=0;

357. begin

358. Put\_line("T5 is started");

359. MatrixInput(MO);

360. DataMonitore.writeAlpha(alpha);

361. SyncMonitore.InputSignal;

362. SyncMonitore.InputtingWait;

363. B5:=DataMonitore.copyB;

364.

365. for i in 4\*H+1..5\*H loop

366. temp:=0;

367. for j in 1..N loop

368. temp:=temp+B5(j)\*MO(i)(j);

369. end loop;

370. T(i):=temp;

371. end loop;

372.

373. SyncMonitore.CalculatingT;

374. SyncMonitore.WaitCalcT;

375.

376. T5:=DataMonitore.copyT;

377. alpha5:=DataMonitore.copyAlpha;

378. MK5:=DataMonitore.copyMK;

379.

380. for i in 4\*H+1..H\*5 loop

381. for j in 1..N loop

382. temp1:=0;

383. for k in 1..N loop

384. temp1:=temp1+T5(j)\*(MT(i)(k)\*MK5(K)(j))+alpha5\*E(i);

385. end loop;

386. end loop;

387. A(i):=temp1;

388. end loop;

389.

390. SyncMonitore.CalculatingA;

391. Put\_line("T5 is ended");

392. end t5;

393.

394. task t6 is

395. end t6;

396. task body t6 is

397. B6,T6:Vector;

398. alpha6:integer;

399. MK6:Matrix;

400. temp,temp1:integer:=0;

401. begin

402. Put\_line("T6 is started");

403.

404. SyncMonitore.InputtingWait;

405. B6:=DataMonitore.copyB;

406.

407. for i in 5\*H+1..6\*H loop

408. temp:=0;

409. for j in 1..N loop

410. temp:=temp+B6(j)\*MO(i)(j);

411. end loop;

412. T(i):=temp;

413. end loop;

414.

415. SyncMonitore.CalculatingT;

416. SyncMonitore.WaitCalcT;

417.

418. T6:=DataMonitore.copyT;

419. alpha6:=DataMonitore.copyAlpha;

420. MK6:=DataMonitore.copyMK;

421.

422. for i in 5\*H+1..H\*6 loop

423. for j in 1..N loop

424. temp1:=0;

425. for k in 1..N loop

426. temp1:=temp1+T6(j)\*(MT(i)(k)\*MK6(k)(j))+alpha6\*E(i);

427. end loop;

428. end loop;

429. A(i):=temp1;

430. end loop;

431.

432. SyncMonitore.CalculatingA;

433. Put\_line("T6 is ended");

434. end t6;

435.

436. task t7 is

437. end t7;

438. task body t7 is

439. B7,T7:Vector;

440. alpha7:integer;

441. MK7:Matrix;

442. temp,temp1:integer:=0;

443. begin

444. Put\_line("T7 is started");

445.

446. SyncMonitore.InputtingWait;

447. B7:=DataMonitore.copyB;

448.

449. for i in 6\*H+1..7\*H loop

450. temp:=0;

451. for j in 1..N loop

452. temp:=temp+B7(j)\*MO(i)(j);

453. end loop;

454. T(i):=temp;

455. end loop;

456.

457. SyncMonitore.CalculatingT;

458. SyncMonitore.WaitCalcT;

459.

460. T7:=DataMonitore.copyT;

461. alpha7:=DataMonitore.copyAlpha;

462. MK7:=DataMonitore.copyMK;

463.

464. for i in 6\*H+1..H\*7 loop

465. for j in 1..N loop

466. temp1:=0;

467. for k in 1..N loop

468. temp1:=temp1+T7(j)\*(MT(i)(k)\*MK7(k)(j))+alpha7\*E(i);

469. end loop;

470. end loop;

471. A(i):=temp1;

472. end loop;

473.

474. SyncMonitore.CalculatingA;

475. Put\_line("T7 is ended");

476. end t7;

477.

478. task t8 is

479. end t8;

480. task body t8 is

481. B8,T8:Vector;

482. alpha8:integer;

483. MK8:Matrix;

484. temp,temp1:integer:=0;

485. begin

486. Put\_line("T8 is started");

487. MatrixInput(MK);

488. MatrixInput(MT);

489. DataMonitore.writeMK(MK);

490. SyncMonitore.InputSignal;

491. SyncMonitore.InputtingWait;

492.

493. B8:=DataMonitore.copyB;

494.

495. for i in 7\*H+1..N loop

496. temp:=0;

497. for j in 1..N loop

498. temp:=temp+B8(j)\*MO(i)(j);

499. end loop;

500. T(i):=temp;

501. end loop;

502.

503. SyncMonitore.CalculatingT;

504. SyncMonitore.WaitCalcT;

505.

506. T8:=DataMonitore.copyT;

507. alpha8:=DataMonitore.copyAlpha;

508. MK8:=DataMonitore.copyMK;

509.

510. for i in 7\*H+1..H\*8 loop

511. for j in 1..N loop

512. temp1:=0;

513. for k in 1..N loop

514. temp1:=temp1+T8(j)\*(MT(i)(K)\*MK8(K)(J))+alpha8\*E(i);

515. end loop;

516. end loop;

517. A(i):=temp1;

518. end loop;

519.

520.

521. SyncMonitore.CalculatingA;

522. Put\_line("T8 is ended");

523. end t8;

524. begin

525. Put\_Line("");

526. end Tasks;

527. begin

528. Put\_line("Main thread is started");

529. StartTime:=Clock;

530. Tasks;

531. Put\_line("Main thread is finished");

532. end Main;

532 lines: No errors

**Додаток А**

/\*

Serhin Dmytro

Java.MPI

IO-22

A=(B\*MO)(MK\*MT)+alpha\*E

\*/

**import** mpi.\*;

**public** **class** Test {

**static** **int** *P*=8;

**static** **int** *N*=100;

**static** **int** *H*=*N*/*P*;

**static** **private** **void** vectorInput(**int**[] X) {

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

X[i] = 1;

}

**static** **private** **void** matrixInput(**int**[][] MX) {

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

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

MX[i][j] = 1;

}

**static** **void** copy(**int**[][]arr,**int**[][]arr2,**int** ii,**int** jj){

**for**(**int** i=ii;i<jj;i++){

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

arr[i][j]=arr2[i][j];

}

}

}

**static** **void** copy(**int**[]a,**int** b[],**int** pos,**int** pos2){

**for**(**int** i=pos;i<pos2;i++){

a[i]=b[i];

}

}

**public** **static** **void** main(String[]args){

MPI.*Init*(args);

**int** rank=MPI.*COMM\_WORLD*.Rank();

**int** size=MPI.*COMM\_WORLD*.Size();

**int** A[]=**new** **int**[*N*];

**int** B[]=**new** **int**[*N*];

**int** E[]=**new** **int**[*N*];

**int** alpha=1;

**int** MO[][]=**new** **int**[*N*][*N*];

**int** MK[][]=**new** **int**[*N*][*N*];

**int** MT[][]=**new** **int**[*N*][*N*];

**int** T[]=**new** **int**[*N*];

**if**(rank==0){

*vectorInput*(B);

*vectorInput*(E);

}

**else** **if**(rank==3){

*matrixInput*(MO);

}

**else** **if**(rank==7){

*matrixInput*(MK);

*matrixInput*(MT);

}

**if**(rank==0){

MPI.*COMM\_WORLD*.Send(B,0,*N*,MPI.*INT*,1,0 );

MPI.*COMM\_WORLD*.Send(E,2\**H*, *N*, MPI.*INT*, 1, 0);

MPI.*COMM\_WORLD*.Recv(MK,0,*N*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Recv(alpha, 0, 0, MPI.*INT*, 1, 0);

MPI.*COMM\_WORLD*.Recv(MO,0,2\**H*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Recv(MT,0,2\**H*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Send(B,0,*N*,MPI.*INT*,4,0 );

MPI.*COMM\_WORLD*.Send(E,4\**H*,5\**H*,MPI.*INT*,4,0);

MPI.*COMM\_WORLD*.Send(MK,0,*N*,MPI.*INT*,4,0);

MPI.*COMM\_WORLD*.Send(alpha,0,0,MPI.*INT*,4,0);

MPI.*COMM\_WORLD*.Send(MO,4\**H*,5\**H*,MPI.*INT*,4,0);

MPI.*COMM\_WORLD*.Send(MT,4\**H*,5\**H*,MPI.*INT*,4,0);

}

**else** **if**(rank==1){

MPI.*COMM\_WORLD*.Recv(B,0,*N*,MPI.*INT*,0,0);

MPI.*COMM\_WORLD*.Recv(E,2\**H*,*N*,MPI.*INT*,0,0);

MPI.*COMM\_WORLD*.Recv(MK,0,*N*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Recv(alpha, 0, 0, MPI.*INT*, 2, 0);

MPI.*COMM\_WORLD*.Recv(MO,0,4\**H*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Recv(MT,0,4\**H*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Send(MK,0,*N*,MPI.*INT*,0,0);

MPI.*COMM\_WORLD*.Send(alpha, 0, 0, MPI.*INT*, 0, 0);

MPI.*COMM\_WORLD*.Send(MO,0,2\**H*,MPI.*INT*,0,0);

MPI.*COMM\_WORLD*.Send(MT,0,2\**H*,MPI.*INT*,0,0);

MPI.*COMM\_WORLD*.Send(B,0,*N*,MPI.*INT*,5,0);

MPI.*COMM\_WORLD*.Send(E, 5\**H*, 6\**H*, MPI.*INT*, 5, 0);

MPI.*COMM\_WORLD*.Send(MK,0,*N*,MPI.*INT*,5,0);

MPI.*COMM\_WORLD*.Send(alpha,0,0,MPI.*INT*,5,0);

MPI.*COMM\_WORLD*.Send(MO,5\**H*,6\**H*,MPI.*INT*,5,0);

MPI.*COMM\_WORLD*.Send(MT,5\**H*,6\**H*,MPI.*INT*,5,0);

MPI.*COMM\_WORLD*.Send(B,0,*N*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Send(E,4\**H*,*N*,MPI.*INT*,2,0);

}

**else** **if**(rank==2){

MPI.*COMM\_WORLD*.Recv(B,0,*N*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Recv(E,4\**H*,*N*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Recv(MK,0,*N*,MPI.*INT*,3,0);

MPI.*COMM\_WORLD*.Recv(alpha, 0, 0, MPI.*INT*, 3, 0);

MPI.*COMM\_WORLD*.Recv(MO,0,6\**H*,MPI.*INT*,3,0);

MPI.*COMM\_WORLD*.Recv(MT,0,6\**H*,MPI.*INT*,3,0);

MPI.*COMM\_WORLD*.Send(MK,0,*N*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Send(alpha, 0, 0, MPI.*INT*, 1, 0);

MPI.*COMM\_WORLD*.Send(MO,0,4\**H*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Send(MT,0,4\**H*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Send(B,0,*N*,MPI.*INT*,6,0);

MPI.*COMM\_WORLD*.Send(E, 6\**H*, 7\**H*, MPI.*INT*, 6, 0);

MPI.*COMM\_WORLD*.Send(MK,0,*N*,MPI.*INT*,6,0);

MPI.*COMM\_WORLD*.Send(alpha,0,0,MPI.*INT*,6,0);

MPI.*COMM\_WORLD*.Send(MO,6\**H*,7\**H*,MPI.*INT*,6,0);

MPI.*COMM\_WORLD*.Send(MT,6\**H*,7\**H*,MPI.*INT*,6,0);

MPI.*COMM\_WORLD*.Send(B,0,*N*,MPI.*INT*,3,0);

MPI.*COMM\_WORLD*.Send(E,6\**H*,*N*,MPI.*INT*,3,0);

}

**else** **if**(rank==3){

MPI.*COMM\_WORLD*.Recv(B,0,*N*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Recv(E,6\**H*,*N*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Recv(MO,0,7\**H*,MPI.*INT*,7,0);

MPI.*COMM\_WORLD*.Recv(alpha,0,0,MPI.*INT*,7,0);

MPI.*COMM\_WORLD*.Send(MK,0,*N*,MPI.*INT*,7,0);

MPI.*COMM\_WORLD*.Send(MT,7\**H*,*N*,MPI.*INT*,7,0);

MPI.*COMM\_WORLD*.Send(B,0,*N*,MPI.*INT*,7,0);

MPI.*COMM\_WORLD*.Send(E,7\**H*,*N*,MPI.*INT*,7,0);

MPI.*COMM\_WORLD*.Send(MK,6\**H*,*N*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Send(alpha, 0, 0, MPI.*INT*, 2, 0);

MPI.*COMM\_WORLD*.Send(MO,0,6\**H*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Send(MT,0,6\**H*,MPI.*INT*,2,0);

}

**else** **if**(rank==4){

MPI.*COMM\_WORLD*.Recv(B,0,*N*,MPI.*INT*,0,0);

MPI.*COMM\_WORLD*.Recv(E,4\**H*,5\**H*,MPI.*INT*,0,0);

MPI.*COMM\_WORLD*.Recv(MK,4\**H*,5\**H*,MPI.*INT*,0,0);

MPI.*COMM\_WORLD*.Recv(alpha, 0, 0, MPI.*INT*, 0, 0);

MPI.*COMM\_WORLD*.Recv(MO,4\**H*,5\**H*,MPI.*INT*,0,0);

MPI.*COMM\_WORLD*.Recv(MT,4\**H*,5\**H*,MPI.*INT*,0,0);

}

**else** **if**(rank==5){

MPI.*COMM\_WORLD*.Recv(B,0,*N*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Recv(E,5\**H*,6\**H*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Recv(MK,5\**H*,6\**H*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Recv(alpha, 0, 0, MPI.*INT*, 1, 0);

MPI.*COMM\_WORLD*.Recv(MO,5\**H*,6\**H*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Recv(MT,5\**H*,6\**H*,MPI.*INT*,1,0);

}

**else** **if**(rank==6){

MPI.*COMM\_WORLD*.Recv(B,0,*N*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Recv(E,6\**H*,7\**H*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Recv(MK,6\**H*,7\**H*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Recv(alpha, 0, 0, MPI.*INT*, 2, 0);

MPI.*COMM\_WORLD*.Recv(MO,6\**H*,7\**H*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Recv(MT,6\**H*,7\**H*,MPI.*INT*,2,0);

}

**else** **if**(rank==7){

MPI.*COMM\_WORLD*.Send(MO,0,7\**H*,MPI.*INT*,3,0);

MPI.*COMM\_WORLD*.Send(alpha,0,0,MPI.*INT*,3,0);

MPI.*COMM\_WORLD*.Recv(B,0,*N*,MPI.*INT*,3,0);

MPI.*COMM\_WORLD*.Recv(E,7\**H*,*N*,MPI.*INT*,3,0);

MPI.*COMM\_WORLD*.Recv(MK,7\**H*,*N*,MPI.*INT*,3,0);

MPI.*COMM\_WORLD*.Recv(MT,7\**H*,*N*,MPI.*INT*,3,0);

}

**int** from = rank\**H*;

**int** to = from+*H*;

**for**(**int** i=from;i<to;i++){

**int** tmp=0;

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

tmp+=B[j]\*MO[i][j];

}

T[i]=tmp;

}

**if**(rank==0){

MPI.*COMM\_WORLD*.Recv(T,4\**H*,5\**H*,MPI.*INT*,4,0);

MPI.*COMM\_WORLD*.Recv(T,2\**H*,*N*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Send(T,0,4\**H*,MPI.*INT*,4,0);

MPI.*COMM\_WORLD*.Send(T,5\**H*,*N*,MPI.*INT*,4,0);

MPI.*COMM\_WORLD*.Send(T,0,2\**H*,MPI.*INT*,1,0);

}

**if**(rank==1){

MPI.*COMM\_WORLD*.Recv(T,5\**H*,6\**H*,MPI.*INT*,5,0);

MPI.*COMM\_WORLD*.Recv(T,0,2\**H*,MPI.*INT*,0,0);

MPI.*COMM\_WORLD*.Recv(T,4\**H*,*N*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Send(T,2\**H*,*N*,MPI.*INT*,0,0);

MPI.*COMM\_WORLD*.Send(T,5\**H*,*N*,MPI.*INT*,4,0);

MPI.*COMM\_WORLD*.Send(T,0,4\**H*,MPI.*INT*,2,0);

}

**if**(rank==2){

MPI.*COMM\_WORLD*.Recv(T,6\**H*,*N*,MPI.*INT*,3,0);

MPI.*COMM\_WORLD*.Recv(T,0,4\**H*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Recv(T,6\**H*,7\**H*,MPI.*INT*,6,0);

MPI.*COMM\_WORLD*.Send(T,4\**H*,*N*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Send(T,0,6\**H*,MPI.*INT*,3,0);

MPI.*COMM\_WORLD*.Send(T,0,7\**H*,MPI.*INT*,6,0);

}

**if**(rank==3){

MPI.*COMM\_WORLD*.Recv(T,0,6\**H*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Recv(T,7\**H*,8,MPI.*INT*,7,0);

MPI.*COMM\_WORLD*.Send(T,6\**H*,*N*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Send(T,0,4\**H*,MPI.*INT*,7,0);

}

**if**(rank==4){

MPI.*COMM\_WORLD*.Recv(T, 0, 7\**H*,MPI.*INT*, 0, 0);

MPI.*COMM\_WORLD*.Send(T,0,4\**H*,MPI.*INT*,0,0);

}

**if**(rank==5){

MPI.*COMM\_WORLD*.Send(T,5\**H*,6\**H*,MPI.*INT*,1,0);

MPI.*COMM\_WORLD*.Recv(T,0,4\**H*,MPI.*INT*,1,0);

}

**if**(rank==6){

MPI.*COMM\_WORLD*.Send(T,6\**H*,7\**H*,MPI.*INT*,2,0);

MPI.*COMM\_WORLD*.Recv(T,0,7\**H*,MPI.*INT*,2,0);

}

**if**(rank==7){

MPI.*COMM\_WORLD*.Send(T,7\**H*,8\**H*,MPI.*INT*,3,0);

MPI.*COMM\_WORLD*.Recv(T,0,7\**H*,MPI.*INT*,3,0);

}

**for**(**int** i=from;i<to;i++){

**int** tmp=0;

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

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

tmp+=T[j]\*(MK[j][k]\*MT[i][j]+alpha\*E[i]);

}

}

A[i]=tmp;}

**if**(rank==7){

MPI.*COMM\_WORLD*.Send(A, 7\**H*, *N*, MPI.*INT*, 3, 0);

}

**if**(rank==3){

MPI.*COMM\_WORLD*.Recv(A,7\**H*,*N*,MPI.*INT*,7,0);

MPI.*COMM\_WORLD*.Send(A,6\**H*,*N*,MPI.*INT*,2,0);

}

**if**(rank==2){

MPI.*COMM\_WORLD*.Recv(A, 6\**H*, *N*, MPI.*INT*, 3,0);

MPI.*COMM\_WORLD*.Recv(A, 5\**H*, 6\**H*, MPI.*INT*, 6, 0);

MPI.*COMM\_WORLD*.Send(A,4\**H*,*N*,MPI.*INT*,1,0);

}

**if**(rank==1){

MPI.*COMM\_WORLD*.Recv(A, 4\**H*, *N*, MPI.*INT*, 2,0);

MPI.*COMM\_WORLD*.Recv(A, 3\**H*, 4\**H*, MPI.*INT*, 5, 0);

MPI.*COMM\_WORLD*.Send(A,4\**H*,*N*,MPI.*INT*,0,0);

}

**if**(rank==0){

MPI.*COMM\_WORLD*.Recv(A,0,*H*,MPI.*INT*,4,0);

MPI.*COMM\_WORLD*.Recv(A,2\**H*,*N*,MPI.*INT*,1,0);

**if**(*N*<18){*vectorOutput*(A);}

}

**if**(rank==6){

MPI.*COMM\_WORLD*.Send(A, 6\**H*, 7\**H*, MPI.*INT*, 2, 0);

}

**if**(rank==5){

MPI.*COMM\_WORLD*.Send(A, 5\**H*, 6\**H*, MPI.*INT*, 1, 0);

}

**if**(rank==4){

MPI.*COMM\_WORLD*.Send(A, 4\**H*, 5\**H*, MPI.*INT*, 2, 0);

}

}

**Додаток С**

Початок

А

Введення

B,E

Копіювання *T1=Т, MK1=МК,* .



Обчислення



Очікування обчислення від Т2-Т8.



Сигнал Т2-Т8 про введення

*Змін.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

1

*Схема алгоритму процесу Т1 в ПКС СП*

*Розроб.*

*Сергін Д.І*

*Перевірив*

*Корочкін О.В.*

*Реценз.*

*Н. Контр.*

*Утверд.*

*Додаток В*

*Алгоритм процесу Т1*

*Літ*

*Листів*

*НТУУ “КПІ” ФІОТ*

Очікування введення від Т5,Т8

Виведення

А

Копіювання

B1=B

Кінець

Обчислення



Сигнал про обчислення Т Т2-Т8

Очікування обчислення від Т2-Т8.



А

*Змін.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

1

*Схема алгоритму процесу Т5 в ПКС СП*

*Розроб.*

*Сергін Д.І*

*Перевірив*

*Корочкін О.В.*

*Реценз.*

*Н. Контр.*

*Утверд.*

*Додаток В*

*Алгоритм процесу Т5*

*Літ*

*Листів*

*НТУУ “КПІ” ФІОТ*

А

Початок

Введення

МО,



Копіювання *T5=Т, MK5=МК,* .



Обчислення



Сигнал Т2-Т4,Т6-Т8 про введення

Кінець

Очікування введення від Т1,Т8

Копіювання

B5=B

Обчислення



Сигнал про обчислення Т Т1-Т4,Т6-Т8

Очікування обчислення від Т1-Т4,Т6-Т8



А

*Змін.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

1

*Схема алгоритму процесу Т8 в ПКС СП*

*Розроб.*

*Сергін Д.І*

*Перевірив*

*Корочкін О.В.*

*Реценз.*

*Н. Контр.*

*Утверд.*

*Додаток В*

*Алгоритм процесу Т8*

*Літ.*

*Листів*

*НТУУ “КПІ” ФІОТ*

А

Початок

Введення

МК, МТ

Копіювання *T8=Т, MK8=МК,* .



Обчислення



Сигнал Т1-Т7 про введення

Кінець

Очікування введення від Т1,Т5

Копіювання

B8=B

Обчислення



Сигнал про обчислення Т Т1-Т7

Очікування обчислення від Т1-Т7



А

А

Початок

Очікування введення від Т1,Т5,Т8

Копіювання *Ti=Т, MKi=МК,* .



Обчислення



Копіювання

Bi=B

Обчислення



Кінець

Сигнал про обчислення Тj, j,j=



Очікування обчислення від Тj, j,j=



А

*Змін.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

1

*Схема алгоритму процесу Тi (i=2,3,4,6,7) в ПКС СП*

*Розроб.*

*Сергін Д.І*

*Перевірив*

*Корочкін О.В.*

*Реценз.*

*Н. Контр.*

*Утверд.*

*Додаток В*

*Алгоритм процесів Тi*

*Літ.*

*Листів*

*НТУУ “КПІ” ФІОТ*

Початок

Кінець

Т8

Т5

Т1

... …

*Змін.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

1

*Схема роботи основної програми*

*Розроб.*

*Сергін Д.І*

*Перевірив*

*Корочкін О.В.*

*Реценз.*

*Н. Контр.*

*Утверд.*

*Додаток В*

*Схема роботи основної програми*

*Літ.*

*Листів*

*НТУУ “КПІ” ФІОТ*