Національний технічний університет України  
«Київський політехнічний інститут»

Кафедра обчислювальної техніки

(повна назва кафедри, циклової комісії)

Київ – 2013 рік

**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-01

напряму підготовки 050102

«Комп’ютерна інженерія»

Корнієнко Дмитро

(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)

Національний технічний університет України  
«Київський політехнічний інститут»

Факультет (інститут) інформатики та обчислювальної техніки

(повна назва)

Кафедра обчислювальної техніки

(повна назва)

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.050102 «Комп’ютерна інженерія»

(шифр і назва)

**Завдання**

на курсову роботу студенту

Корнієнко Дмитро Ігорович

(прізвище, ім’я, по батькові)

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

(вчене звання, науковий ступінь, прізвище та ініціали)

1. Строк подання студентом роботи 11 травня 2013 р.
2. Вихідні дані для роботи

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

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

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

1. Перелік графічного матеріалу:

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

1. Дата видачі завдання \_\_\_\_\_\_\_\_\_\_\_\_\_

Календарний план

|  |  |  |
| --- | --- | --- |
| № з/п | Назва етапів виконання КР | Строк виконання етапів КР |
|  | Виконання огляду для розділу 1 | 28.03.2013 |
|  | Розробка паралельного алгоритму рішення задачі | 01.04.2013 |
|  | Розробка алгоритмів процесів | 06.04.2013 |
|  | Розробка схем взаємодії процесів | 13.04.2013 |
|  | Розробка програм | 20.04.2013 |
|  | Тестування програм | 30.04.2013 |
|  | Оформлення КР | 10.05.2013 |
|  | Захист КР | 18.05.2013 |

Студент

(підпис) (прізвище та ініціали)

Керівник роботи

**Технічне завдання на курсову роботу**

1 Область застосування

Розроблюване програмне забезпечення може бути застосоване в області паралельних математичних обчислень для масштабованих комп'ютерних систем з загальної та локальної пам'яттю.

2 Підстави для розробки

Підставою для розробки служить ТЗ на курсову роботу.

3 Мета курсової роботи

Метою даної роботи є закріплення отриманих знань з дисципліни "Паралельні та розподілені обчислення", а також отримання навичок і досвіду в розробці програмного забезпечення для комп'ютерних систем з різною структурною організацією.

4 Призначення роботи

Розробка програмного забезпечення (ПО) для ПКС з загальної та локальної пам'яттю. Проведення досліджень ефективності ПЗ.

5 Вихідні дані і виконання роботи

5.1. Завдання для розділу 1: Огляд засобів роботи з процесами у бібліотеці MPI

5.2 Завдання для розділів 2 і 3: Математична задача:

A = B\*MO+C\*(MR\*MX). Розглядаються вектора і матриці розмірності N.

5.3. Структура паралельної комп'ютерної системи з загальної пам'яттю (мал.1):

Спільна пам’ять

1

3

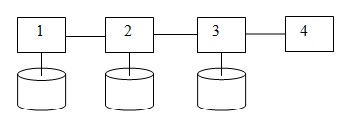
2

4

A, B, C MO,MX MR

Мал.1 Структура ПКС ОП

5.4 Структура паралельної комп'ютерної системи з локальною пам'яттю (мал.2):



A, B, C MO,MX MR

Мал.2 Структура ПКС ЛП

5.5. Мови програмування і засоби організації взаємодії процесів: мова Java і механізм моніторів для розділу 2, бібліотека MPI - для розділу 3.

У КР необхідно: розробити алгоритм рішення заданого математичної задачі і дослідити його паралельні властивості для N і Р, де N - розмірність матриць, Р - кількість процесорів.

Виконати розробку програми рішення заданого математичного виразу для заданих структур ПКС на заданих мовах програмування. Описати структуру взаємодії задач.

Виконати дослідження ефективності ПО в залежності від значень Р і N. При цьому підрахувати час виконання математичної задачі на різній кількості процесорів. Визначити значення коефіцієнтів прискорення (КП) та ефективності (КЕ).

6 Вимоги до складу і параметрів технічних засобів

Тестування ПО виконується за використання багатоядерної ПКС

7 Вимоги до інформаційної та програмної сумісності

Програма повинна працювати під управлінням ОС Windows.

8 Спеціальні вимоги

Тестування виконується на 6-х ядерної системі.

9 Етапи та стадії розробки

- Узгодження технічного завдання

- Виконання курсової роботи.

- Захист курсової роботи.

21. 04. 2013

**Технічне завдання на курсову роботу**

1 Область застосування

Розроблюване програмне забезпечення може бути застосоване в області паралельних математичних обчислень для масштабованих комп'ютерних систем з загальної та локальної пам'яттю.

2 Підстави для розробки

Підставою для розробки служить ТЗ на курсову роботу.

3 Мета курсової роботи

Метою даної роботи є закріплення отриманих знань з дисципліни "Паралельні та розподілені обчислення", а також отримання навичок і досвіду в розробці програмного забезпечення для комп'ютерних систем з різною структурною організацією.

4 Призначення роботи

Розробка програмного забезпечення (ПО) для ПКС з загальної та локальної пам'яттю. Проведення досліджень ефективності ПЗ.

5 Вихідні дані і виконання роботи

5.1. Завдання для розділу 1: Огляд засобів роботи з процесами у бібліотеці MPI

5.2 Завдання для розділів 2 і 3: Математична задача:

A = B\*MO+C\*(MR\*MX). Розглядаються вектора і матриці розмірності N.

5.3. Структура паралельної комп'ютерної системи з загальної пам'яттю (мал.1):

Спільна пам’ять

1

3

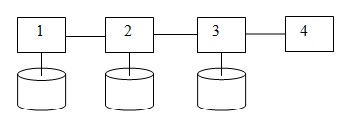
2

4

A, B, C MO,MX MR

Мал.1 Структура ПКС ОП

5.4 Структура паралельної комп'ютерної системи з локальною пам'яттю (мал.2):



A, B, C MO,MX MR

Мал.2 Структура ПКС ЛП

5.5. Мови програмування і засоби організації взаємодії процесів: мова Java і механізм моніторів для розділу 2, бібліотека MPI - для розділу 3.

У КР необхідно: розробити алгоритм рішення заданого математичної задачі і дослідити його паралельні властивості для N і Р, де N - розмірність матриць, Р - кількість процесорів.

Виконати розробку програми рішення заданого математичного виразу для заданих структур ПКС на заданих мовах програмування. Описати структуру взаємодії задач.

Виконати дослідження ефективності ПО в залежності від значень Р і N. При цьому підрахувати час виконання математичної задачі на різній кількості процесорів. Визначити значення коефіцієнтів прискорення (КП) та ефективності (КЕ).

6 Вимоги до складу і параметрів технічних засобів

Тестування ПО виконується за використання багатоядерної ПКС

7 Вимоги до інформаційної та програмної сумісності

Програма повинна працювати під управлінням ОС Windows.

8 Спеціальні вимоги

Тестування виконується на 6-х ядерної системі.

9 Етапи та стадії розробки

- Узгодження технічного завдання

- Виконання курсової роботи.

- Захист курсової роботи.

21. 04. 2013

**Розділ 2. Розробка ПЗ для ПКС з СП**

У цьому розділі буде розглянуто та проаналізовано поставлену математичну задачу, розроблено алгоритм рішення та схему взаємодії потоків для системі з спільною пам’яттю, а на основі вищезазначених досліджень буде створена паралельна програма.

2.1 Рішення початкової математичної задачі

Матаматична задача, для якої необхідно створити паралельну програму: A=B\*MO + C\*(MR\*MX).

Перш за все розроблюється математичний алгоритм, аналізується наявність спільних ресурсів.

AH= B\*MOH +C\*(MR\*MXH)

Спільні ресурси: B, C, MR

Наступним етапом є створення алгоритму для кожного паралельного процесу із зазначенням точок синхронізації та критичних ділянок.

|  |  |
| --- | --- |
| T1 | TC |
| 1. Введення MX, МТ |  |
| 2. Сигнал Т2, T3, T4 – введення закінчено. | S2,3,4-1 |
| 3. Чекати сигналу від Т2, T3 про закінчення введення. | W2,3-1 |
| 4. Копія B1 = B, C1 = C, MR1 = MR | КУ |
| 5. Рахунок AH = B1 \* MOH + С1 \*(MR1 \* MXH) |  |
| 6. Чекати сигналу від Т1, Т2, Т4 о закінченні рахунку. | W1,2,4-3 |
| 7. Вивід А |  |

|  |  |
| --- | --- |
| T2 | TC |
| 1. Введення MЕ |  |
| 2. Сигнал Т1, T3, T4 – введення закінчено. | S1,3,4-1 |
| 3. Чекати сигналу від Т1, T3 про закінчення введення. | W1,3-1 |
| 4. Копія B2 = B, C2 = C, RM2 = MR | КУ |
| 5. Рахунок AH = B2 \* MOH + С2 \*(MR2 \* MXH) |  |
| 6. Сигнал Т1 о закінченні рахунку AH. | S1-3 |

|  |  |
| --- | --- |
| T3 | TC |
| 1. Ввод B |  |
| 2. Сигнал Т2, T1, T4 – введення завершене. | S2,1,4-1 |
| 3. Чекати сигналу від Т2, T1 про закінчення введення. | W2,1-1 |
| 4. Копія B3 = B, C3 = C, MR3 = MR | КУ |
| 5. Рахунок AH = B3 \* MOH + С3 \*(MR3 \* MXH) |  |
| 6. Сигнал Т1 о закінченні рахунку AH. | S1-3 |

|  |  |
| --- | --- |
| T4 | TC |
| 1. Чекати сигналу від Т2, T3, T1 о завершення ведення. | W2,3,1-1 |
| 2. Копия B4 = B, C4 = C, MR4 = MR | КУ |
| 3. Счёт AH = B4 \* MOH + С4 \*(MR4 \* MXH) |  |
| 4. Сигнал Т1 о завершении счета. | S1-3 |

На підставі даного алгоритму будується схема взаємодії процесів, представлена ​​в додатку Г «Алгоритми процесів в програмі для ПКС з СП». Як засіб синхронізації обраний механізм монітора, зображенного на рисунку 2.1, в програмі створений клас з даною назвою, який реалізує виконання задач взаємного виключення і синхронізації, представлених вище в алгоритмі.



Рисунок 2.1 – Монітор

Використовуються такі методи:

- WaitForInput - очікування закінчення введення Т1-T4 (синхронізація по вводу). Прапор F спочатку встановлений в 0. При закінченні введення якихось із цих потоків, виконується операція F1 + + (в методі InputSignal). Якщо F = 3, все потоки повідомляються про це і отримують можливість доступу до даних і до подальшого виконання програми.

- WaitForCalc - очікування закінчення рахунку (синхронізація з виведення). Прапор F спочатку встановлений в 0. При закінченні рахунки яким або з потоків, виконується операція F + + (в методі CalcSignal). Якщо F1 = 4, потік (Р) отримує повідомлення про це і допускається до подальшого виконання програми.

- InputSignal - повідомлення від потоку про завершення їм введення даних. F1 + +.

- CalcSignal - повідомлення від потоку про завершення їм рахунки. F2+ +.

Лістинг програми розміщений у додатку Ж «Лістинг програми для ПКС з СП».

2.2 Дослідження ефективності розробленого ПЗ на реальній Р-ядерної ПКС

Для дослідження необхідно визначити час виконання розробленої програми в реальній ПКС. Тому при тестуванні послідовно використовуються 1, 2, 4, 6 процесорів, для яких визначаються час виконання Програми 1. При цьому встановлюється декілька значень розмірності векторів (матриць) N = 1000, 2000, 3000. В таблиці 2.1 відображаються значення часу для різних N та Р.

Таблиця 2.1 - Час виконання програми в ПКС з СП (в мс)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N//P | P=1 | P=2 | P=3 | P=4 |
| 800 | 8736 | 4879 | 3405 | 2544 |
| 1600 | 95833 | 53906 | 40749 | 32029 |
| 2400 | 444356 | 240565 | 180948 | 132697 |

Базуючись на отриманих результатах, вираховуємо значення коефіцієнтів прискорення . У таблиці 2.2 приведено результати цих розрахунків.

Таблиця 2.2. Значения Кп для ПКС з СП

|  |  |  |  |
| --- | --- | --- | --- |
| N//P | P=2 | P=3 | P=4 |
| 800 | 1.79 | 2.56 | 3.43 |
| 1600 | 1.77 | 2.35 | 2.99 |
| 2400 | 1.85 | 2.46 | 3.35 |

На основі отриманих коефіцієнтів прискориння будується графік змінення КП в залежності від Р та N. Графік наведено на рис. 2.2. На осі абцис разташована розмірність векторів та матриць, ординатами зазначено коефіцієнт прискорення. Для позначення результатів різних багатопроцесорних систем використовуються різні типи ліній та маркеров.

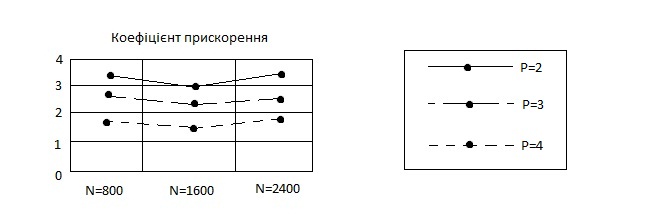


Рисунок 2.2 - Графік залежності коефіцієнта прискорення від розмірності матриць і векторів для ПКС з СП.

Розрахуємо коефіцієнти ефективності та заповнимо їми таблицю 2.3. Для розрахунку використовуються формули  .

Таблиця 2.3 - Значення Ке для програми ПКС з СП

|  |  |  |  |
| --- | --- | --- | --- |
| N//P | P=2 | P=3 | P=4 |
| 800 | 0.895 | 0.855 | 0.858 |
| 1600 | 0.889 | 0.784 | 0.748 |
| 2400 | 0.923 | 0.819 | 0.837 |

На основі отриманих коефіцієнтів ефективності будується графік змінення Ке в залежності від Р та N. Графік наведено на рис. 2.3. На осі абцис разташована розмірність векторів та матриць, ординатами зазначено коефіцієнт ефективності. Для позначення результатів різних багатопроцесорних систем використовуються різні типи ліній та маркеров.

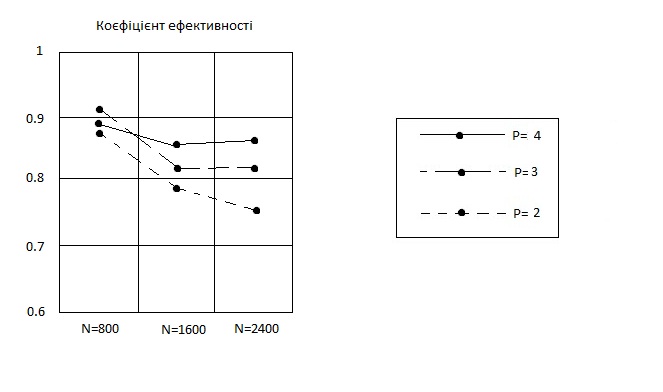


Рисунок 2.3 - Графік залежності коефіцієнта прискорення від розмірності матриць і векторів для ПКС з СП.

Висновки до розділу 2

1. У даному розділі досліджені результати тестування паралельної програми для системи з загальною пам’яттю, написаної на мові Java. Тестування провдилось для 2, 3 та 4 потоків. Для розрахунку коефіцієнтів прискорення та ефективності була протестована окремо створена послідовна програма.

2. Коефіцієнт прискорення приймає значення у проміжку від 1.77 до 3.43. Найвищі значення цього коефіцієнту приходяться на систему з чотирьма потоками, найменше прискорення отримане для системи з двома потоками. Характер графіків коефіцієнтів прискорення однаковий для систем з 2, 3 та 4 ядрами.

3. Коефіцієнт ефективності приймає значення у проміжку від 0,923до 0,748. Найвище значення цього коефіцієнту отримане на тесті з параметрами Р = 3, N = 800; найнижче – P = 2, N = 2400.

4. Результати дослідження мають похибку, зумовлену тим, що процесори виділяються операційною системою не на монопольне використання, тобто, процесорний час може бути в будь-який час передано сторонній програмі. Чим менший час виконання програми, тим більша вирогідність виникнення досить значущої похибки.

**Розділ 3. Розробка ПЗ для ПКС з ЛП**

У цьому розділі буде розглянуто та проаналізовано поставлену математичну задачу, розроблено алгоритм рішення та схему взаємодії задач, а на основі вищезазначених досліджень буде створена паралельна програма для системи з локальною пам’яттю.

Математична задача співпадає з уже розглянутою у розділі 2.1, тому розділ «Рішення початкової математичної задачі» пропущений.

3.2 Розробка алгоритмів процесів.

Програмне забезпечення є масштабованим, тобто працює на системі з будь‑якою кількістю процесорів. Тому написаний алгоритм єдиний для всіх задач.

Задача T0

1. **Введення B,C**
2. **Прийняти** MO, MX від задачі T1.
3. **Передати** задачі T1 B, C
4. **Прийняти** MR від задачі T1.
5. **Обчислення** AH= B\*MOH +C\*(MR\*MXH).
6. **Прийняти** результат обчислення А3Н від Т1.
7. **Виведення** результату А.

Задача T1

1. **Прийняти** B, C від задачі T0.
2. **Передати** задачі T0 MOH, MXH.
3. **Передати** задачі T2 MO2H, MX2H.
4. **Передати** MR задачі T0.
5. **Прийняти** MR від задачі T2.
6. **Обчислення** AH= B\*MOH +C\*(MR\*MXH).
7. **Передати** результат A3H задачі Т0
8. **Прийняти** резултат А2Н від задачі Т2

Задача T2

1. **Прийняти** B, C від задачі T1.
2. **Передати** B, C задачі T3.
3. **Передати** задачі T3 MOH, MXH.
4. **Прийняти від задачі T1 MO2H, MX2H.**
5. **Передати** задачі T1 MR2H.
6. **Передати** MRH задачі T3.
7. **Обчислення** AH= B\*MOH +C\*(MR\*MXH).
8. **Передати** результат A2H задачі Т1
9. **Прийняти** резултат АН від задачі Т3.

Задача T3

1. **Прийняти** B, C від задачі T2.
2. **Прийняти** MOH, MXH від задачі T2.
3. **Прийняти** MR від задачі T2.
4. **Обчислення** AH= B\*MOH +C\*(MR\*MXH).
5. **Передати** результат AH задачі Т2.

3.3 Розробка схеми взаємодії процесів

Схема взаємодії задач приведена на рисунку 3.1.



Рисунок 3.1 – Схема взаємодії задач в програмі ПРГ2

3.4 Розробка програми

Програма для ПКС ЛП складена на мові C++ з використанням бібліотеки MPI та містить чотири модулі: основний Coursework2.cpp та допоміжний data.cpp.

Основний модуль містить точку входу в програму main та реалізує алгоритм процесів, описаний в розділі 3.2. Для передачі і приймання використовувались функції MPI\_Send, MPI\_Recv.

Модуль data.cpp зберігає процедури, за допомогою яких формуються основні обчислення у програмі ПРГ2.

Лістинг програми наведено у додатку Ж.

3.6 Дослідження ефективності розробленого ПЗ на реальній Р-ядерної ПКС

Для дослідження необхідно визначити час виконання розробленої програми в реальній ПКС. Тому при тестуванні послідовно використовуються 1, 2, 4, 6 процесорів, для яких визначаються час виконання Програми 1. При цьому встановлюється декілька значень розмірності векторів (матриць) N = 1000, 2000, 3000. В таблиці 2.1 відображаються значення часу для різних N та Р.

Таблиця 3.1 - Час виконання програми в ПКС з СП (в мс)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N//P | P=1 | P=2 | P=3 | P=4 |
| 800 | 1782 | 1132 | 848 | 831 |
| 1600 | 12152 | 6411 | 4529 | 4018 |
| 2400 | 68383 | 35013 | 22786 | 20344 |

Базуючись на отриманих результатах, вираховуємо значення коефіцієнтів прискорення . У таблиці 2.2 приведено результати цих розрахунків.

Таблиця 3.2. Значения Кп для ПКС з СП

|  |  |  |  |
| --- | --- | --- | --- |
| N//P | P=2 | P=3 | P=4 |
| 800 | 1.57 | 2.11 | 2.14 |
| 1600 | 1.89 | 2.68 | 3.02 |
| 2400 | 1.95 | 3.01 | 3.36 |

На основі отриманих коефіцієнтів прискориння будується графік змінення КП в залежності від Р та N. Графік наведено на рис. 2.2. На осі абцис разташована розмірність векторів та матриць, ординатами зазначено коефіцієнт прискорення. Для позначення результатів різних багатопроцесорних систем використовуються різні типи ліній та маркеров.

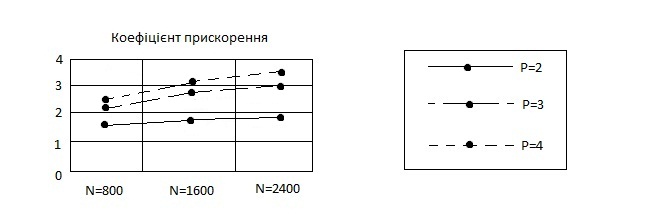


Рисунок 3.2 - Графік залежності коефіцієнта прискорення від розмірності матриць і векторів для ПКС з СП.

Розрахуємо коефіцієнти ефективності та заповнимо їми таблицю 2.3. Для розрахунку використовуються формули  .

Таблиця 3.3 - Значення Ке для програми ПКС з СП

|  |  |  |  |
| --- | --- | --- | --- |
| N//P | P=2 | P=3 | P=4 |
| 800 | 0.787 | 0.701 | 0.611 |
| 1600 | 0.947 | 0.894 | 0.694 |
| 2400 | 0.976 | 0.914 | 0.840 |

На основі отриманих коефіцієнтів ефективності будується графік змінення Ке в залежності від Р та N. Графік наведено на рис. 2.3. На осі абцис разташована розмірність векторів та матриць, ординатами зазначено коефіцієнт ефективності. Для позначення результатів різних багатопроцесорних систем використовуються різні типи ліній та маркеров.

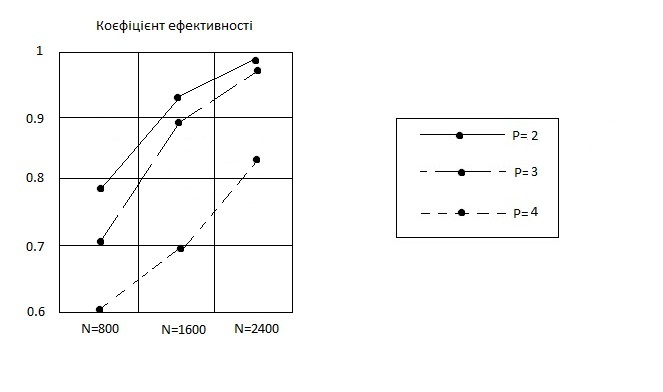


Рисунок 3.3 - Графік залежності коефіцієнта прискорення від розмірності матриць і векторів для ПКС з СП.

3.7 Висновки до розділу 3

1. У даному розділі досліджені результати тестування паралельної програми для системи з загальною пам’яттю, написаної на мові C++ з використанням бібліотеки MPI. Тестування провдилось для 2, 3 та 4 потоків. Для розрахунку коефіцієнтів прискорення та ефективності була протестована окремо створена послідовна програма.

2. Коефіцієнт прискорення приймає значення у проміжку від 1.57 до 3.01. Найвищі значення цього коефіцієнту приходяться на систему з чотирьма потоками, найменше прискорення отримане для системи з двома потоками. Характер графіків коефіцієнтів прискорення однаковий для систем з 2, 3 та 4 ядрами.

3. Коефіцієнт ефективності приймає значення у проміжку від 0,911до 0,976. Найвище значення цього коефіцієнту отримане на тесті з параметрами Р = 2, N = 2400; найнижче – P = 4, N = 800.

4. Результати дослідження мають похибку, зумовлену тим, що процесори виділяються операційною системою не на монопольне використання, тобто, процесорний час може бути в будь-який час передано сторонній програмі. Чим менший час виконання програми, тим більша вирогідність виникнення досить значущої похибки.

# РОЗДІЛ 4. Основні висновки та результати роботи

1. У даному курсовому проекті було розглянуто Огляд засобів роботи з процесами у бібліотеці MPI.

2. Тестування програм для системи як з загальною пам’яттю, так і з локальною, показало передбачувані результати, а саме, зростання прискорення виконання програми при збільшенні кількості процесорів (для такої самої розмірності векторів та матриць).

3. Більш ефективною виявилась програма для системи з локальною пам’яттю, реалізована на мові C++ з використанням біліотеки MPI. Це може бути зумовлено декількома причинами. Процесори у системі з ЛП утворюють зірку, тому передача даних, що проходить паралельно у декількох напрямках зірки, на відносно невеликій кількості процесорів займає приблизно стільки ж часу, скільки отримання захищених монітором даних у програмі для системи з ЗП.

4. При тестуванні мала місце похибка, зумовлена непередбачуваним для користувача розподіленням процесорного часу операційною системою. Похибка зменшується при збільшенні розмірів вхідних даних, тобто, залежить від часу виконання програми.

ДОДАТКИ

Приложение А. Структурная схема ПКС с ОП

Приложение Б. Структурная схема ПКС с ЛП

Приложение В. Алгоритм основной программы для ПКС с ОП

Приложение Г. Алгоритмы процессов в программе для ПКС с ОП

Приложение Д. Алгоритм основной программы для ПКС с ЛП

Приложение Е. Алгоритмы процессов в программе для ПКС с ЛП

Приложение Ж. Листинг программы для ПКС с ОП

Приложение З. Листинг программы для ПКС с ЛП.

*Спільна пам’ять*

*Процесор 1*

*Процесор 3*

*ПВВ*

*ПВВ*

Процесор 2

ПВВ

Процесор 4

ПВВ

A, B, C MO, MX MR

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

1

ІАЛЦ 462637.001

Розроб.

Корнієнко Д. І.

Перевір.

Корочкін О.В..

Реценз.

Н. Контр.

Затверд.

Додаток А. Структурна схема ПКС з СП

Літ.

Акрушів

1

НТУУ «КПІ»



Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

1

ІАЛЦ 462637.002

Розроб.

Корнієнко Д. І.

Перевір.

Корочкін О.В..

Реценз.

Н. Контр.

Затверд.

Додаток Б. Структурна схема ПКС з ЛП

Літ.

Акрушів

1

НТУУ «КПІ»



Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

1

ІАЛЦ 462637.003

Розроб.

Корнієнко Д. І.

Перевір.

Корочкін О.В..

Реценз.

Н. Контр.

Затверд.

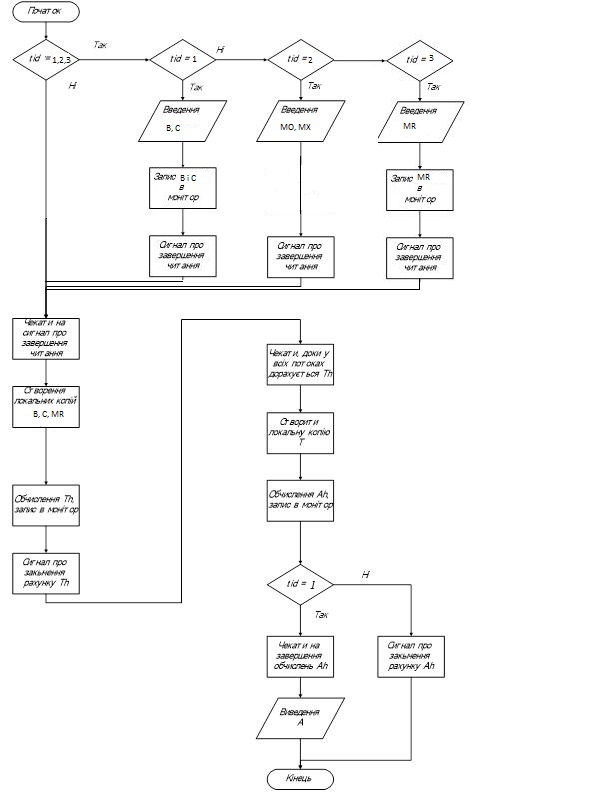
Додаток В. Алгоритм програми ПКС з СП

Літ.

Акрушів

1

НТУУ «КПІ»



Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

1

ІАЛЦ 462637.004

Розроб.

Корнієнко Д. І.

Перевір.

Корочкін О.В..

Реценз.

Н. Контр.

Затверд.

Додаток Г. Алгоритм процесу ПКС з СП

Літ.

Акрушів

1

НТУУ «КПІ»



Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

1

ІАЛЦ 462637.005

Розроб.

Корнієнко Д. І.

Перевір.

Корочкін О.В..

Реценз.

Н. Контр.

Затверд.

Додаток Д. Алгоритм програми ПКС з ЛП

Літ.

Акрушів

1

НТУУ «КПІ»



Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

1

ІАЛЦ 462637.006

Розроб.

Корнієнко Д.

Перевір.

Корочкін О.В..

Реценз.

Н. Контр.

Затверд.

Додаток Е. Алгоритм процесів ПКС з ЛП

Літ.

Акрушів

1

НТУУ «КПІ»

Додаток Ж. Листинг програми ПКС СП

//PRO course work Java

//Kornienko Dmytro IO-01

//A = B\*MO+C\*(MR\*MX)

//26.03.2012

**public** **class** Exec {

**static** **long** *startTime* = System.*currentTimeMillis*();

**public** **static** **int** *N* = 800;

**public** **static** **int** *P* = 4;

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

**public** **static** Vector *A* = **new** Vector(*N*);

**public** **static** Vector *B* = **new** Vector(*N*);

**public** **static** Vector *C* = **new** Vector(*N*);

**public** **static** Matrix *MO* = **new** Matrix(*N*);

**public** **static** Matrix *MX* = **new** Matrix(*N*);

**public** **static** Matrix *MR* = **new** Matrix(*N*);

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

System.*out*.println("Lab 5 started");

InputSync inputSync = **new** InputSync();

CalcSync calcSync = **new** CalcSync();

Resources resources = **new** Resources();

Thread t1 = **new** Thread(**new** T1(inputSync, calcSync, resources));

Thread t2 = **new** Thread(**new** T2(inputSync, calcSync, resources));

Thread t3 = **new** Thread(**new** T3(inputSync, calcSync, resources));

Thread t4 = **new** Thread(**new** T4(inputSync, calcSync, resources));

t1.start();

t2.start();

t3.start();

t4.start();

**try** {

t1.join();

t2.join();

t3.join();

t4.join();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.*out*.println("Lab 5 finished");

**long** timeSpent = System.*currentTimeMillis*() - *startTime*;

System.*out*.println("программа выполнялась " + timeSpent + " миллисекунд");

}

}

**package** course;

**public** **class** CalcSync {

**private** **static** **int** *F1* = 0;

**public** **synchronized** **void** calcSignal() {

*F1*++;

**if** (*F1* >= 3) {

notifyAll();

}

}

**public** **synchronized** **void** waitForCalc() {

**try** {

**if** (*F1* < 3) {

wait();

}

Operations.*outputVector*(Exec.*A*);

} **catch**(Exception e){

e.printStackTrace();

}

}

}

**package** course;

**public** **class** InputSync {

**private** **static** **int** *F* = 0;

**public** **synchronized** **void** inputSignal() {

*F*++;

**if** (*F* >= 3) {

notifyAll();

}

}

**public** **synchronized** **void** waitForInput() {

**try** {

**if** (*F* < 3) {

wait();

}

} **catch**(Exception e){

e.printStackTrace();

}

}

}

**package** course;

**public** **class** Resources {

**private** Vector B = Operations.*inputVector*(Exec.*N*, 1);

**private** Vector C = Operations.*inputVector*(Exec.*N*, 1);

**private** Matrix MR = Operations.*inputMatrix*(Exec.*N*, 1);

**public** **synchronized** Vector getB() {

**return** B;

}

**public** **synchronized** Vector getC() {

**return** C;

}

**public** **synchronized** Matrix getMR() {

**return** MR;

}

**public** **synchronized** **void** setB(Vector B) {

**this**.B = B;

}

**public** **synchronized** **void** setC(Vector C) {

**this**.C = C;

}

**public** **synchronized** **void** setMR(Matrix MR) {

**this**.MR = MR;

}

}

**package** course;

/\*\*

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

\* ЗАДАЧА 1 \*

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

\*/

**public** **class** T1 **implements** Runnable {

**private** **int** id = 1;

**private** InputSync inputSync;

**private** Resources resources;

**private** CalcSync calcSync;

**public** T1(InputSync inputSync, CalcSync calcSync, Resources resources) {

**this**.inputSync = inputSync;

**this**.resources = resources;

**this**.calcSync = calcSync;

}

@Override

**public** **void** run() {

System.*out*.println("Task T1 started");

// 1. Введення B, C

Vector B1 = Operations.*inputVector*(Exec.*N*, 1);

resources.setB(B1);

Vector C1 = Operations.*inputVector*(Exec.*N*, 1);

resources.setC(C1);

// 2. Сигнал про завершення вводу B, C

inputSync.inputSignal();

// 3. Чекати на завершення вводу в інших процесах

inputSync.waitForInput();

// 4. Копіювати MR1 = MR

Matrix MR1 = resources.getMR();

// 5. Обчислення AH = B1 \* (MOH \* MX1 + ME1 \* MTH)

Operations.*calculation*(B1, C1, MR1, id);

// 6. Сигнал про завершення обчислення AH

calcSync.calcSignal();

System.*out*.println("Task T1 finished");

}

}

**package** course;

/\*\*

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

\* ЗАДАЧА 2 \*

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

\*/

**public** **class** T2 **implements** Runnable {

**private** **final** **int** id = 2;

**private** InputSync inputSync;

**private** Resources resources;

**private** CalcSync calcSync;

**public** T2(InputSync inputSync, CalcSync calcSync, Resources resources) {

**this**.inputSync = inputSync;

**this**.resources = resources;

**this**.calcSync = calcSync;

}

@Override

**public** **void** run() {

System.*out*.println("Task T2 started");

// 1. Введення MO, MX

Exec.*MO* = Operations.*inputMatrix*(Exec.*N*, 1);

Exec.*MX* = Operations.*inputMatrix*(Exec.*N*, 1);

// 2. Сигнал про завершення вводу MO, MX

inputSync.inputSignal();

// 3. Чекати на завершення вводу в інших процесах

inputSync.waitForInput();

// 4. Копіювати B2 = B, C2 = C, MR2 = MR

Vector B2 = resources.getB();

Vector C2 = resources.getC();

Matrix MR2 = resources.getMR();

// 5. Обчислення AH = B2 \* (MOH \* MX2 + ME2 \* MTH)

Operations.*calculation*(B2, C2, MR2, id);

// 6. Сигнал про завершення обчислення AH

calcSync.calcSignal();

System.*out*.println("Task T2 finished");

}

}

**package** course;

/\*\*

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

\* ЗАДАЧА 3 \*

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

\*/

**public** **class** T3 **implements** Runnable {

**private** **final** **int** id = 3;

**private** InputSync inputSync;

**private** Resources resources;

**private** CalcSync calcSync;

**public** T3(InputSync inputSync, CalcSync calcSync, Resources resources) {

**this**.inputSync = inputSync;

**this**.resources = resources;

**this**.calcSync = calcSync;

}

@Override

**public** **void** run() {

System.*out*.println("Task T3 started");

// 1. Введення MR

Matrix MR3 = Operations.*inputMatrix*(Exec.*N*, 1);

resources.setMR(MR3);

// 2. Сигнал про завершення вводу MR

inputSync.inputSignal();

// 3. Чекати на завершення вводу в інших процесах

inputSync.waitForInput();

// 4. Копіювати B3 = B, C3 = C3

Vector B3 = resources.getB();

Vector C3 = resources.getC();

// 5. Обчислення AH = B3 \* (MOH \* MX3 + ME3 \* MTH)

Operations.*calculation*(B3, C3, MR3, id);

// 6. Чекати на завершення обчислень AH в інших процесах

calcSync.waitForCalc();

System.*out*.println("Task T3 finished");

}

}

**package** course;

/\*\*

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

\* ЗАДАЧА 4 \*

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

\*/

**public** **class** T4 **implements** Runnable {

**private** **final** **int** id = 4;

**private** InputSync inputSync;

**private** Resources resources;

**private** CalcSync calcSync;

**public** T4(InputSync inputSync, CalcSync calcSync, Resources resources) {

**this**.inputSync = inputSync;

**this**.resources = resources;

**this**.calcSync = calcSync;

}

@Override

**public** **void** run() {

System.*out*.println("Task T4 started");

// 1. Чекати на завершення вводу в інших процесах

inputSync.waitForInput();

// 2. Копіювати B4 = B, C4 = C, MR4 = MR

Vector B4 = resources.getB();

Vector C4 = resources.getC();

Matrix MR4 = resources.getMR();

// 3. Обчислення AH = B4 \* (MOH \* MX4 + ME4 \* MTH)

Operations.*calculation*(B4, C4, MR4, id);

// 4. Сигнал про завершення обчислення AH

calcSync.calcSignal();

System.*out*.println("Task T4 finished");

}

}

**package** course;

**public** **class** Vector {

**private** **int**[] array;

**public** Vector(**int** n) {

array = **new** **int**[n];

}

**public** **void** set(**int** index, **int** value) {

array[index] = value;

}

**public** **int** get(**int** index) {

**return** array[index];

}

**public** **int** size() {

**return** array.length;

}

**public** String toString() {

String res = "";

**for** (**int** i = 0; i < array.length; i++){

res += " " + array[i];

}

res += "\n";

**return** res;

}

}

**package** course;

**public** **class** Matrix {

**private** Vector[] array;

**public** Matrix(**int** n) {

array = **new** Vector[n];

**for** (**int** i = 0; i < array.length; i++){

array[i] = **new** Vector(n);

}

}

**public** **void** set(**int** n, **int** m, **int** val) {

array[n].set(m, val);

}

**public** **int** get(**int** n, **int** m) {

**return** array[n].get(m);

}

**public** Vector get(**int** index) {

**return** array[index];

}

**public** **int** size() {

**return** array.length;

}

**public** String toString() {

String res = "";

**for** (**int** i = 0; i < array.length; i++){

res += array[i].toString();

}

**return** res;

}

}

**package** course;

/\*\*

\* Class with matrix and vector operations

\*/

**public** **class** Operations {

**public** **static** Vector inputVector(**int** n, **int** value) {

Vector vector = **new** Vector(n);

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

vector.set(i, value);

}

**return** vector;

}

**public** **static** Matrix inputMatrix(**int** n, **int** value) {

Matrix matrix = **new** Matrix(n);

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

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

matrix.set(i, j, value);

}

}

**return** matrix;

}

**public** **static** **void** outputVector(Vector vector) {

**if**(vector.size() <= 12) {

System.*out*.print(vector.toString());

}

}

**public** **static** **void** outputMatrix(Matrix matrix) {

**if**(matrix.size() <= 12) {

System.*out*.print(matrix.toString());

}

}

**protected** **static** Vector add(**final** Vector left, **final** Vector right,

**final** **int** id) {

**int** l = (id - 1) \* Exec.*H*;

**int** r = id \* Exec.*H*;

**if**(id==4){

r = Exec.*N*;

}

Vector result = **new** Vector(left.size());

**for**(**int** i = l; i < r; i++) {

result.set(i, left.get(i) + right.get(i));

}

**return** result;

}

**protected** **static** Vector mult(**final** Vector left, **final** Matrix right,

**final** **int** id) {

**int** l = (id - 1) \* Exec.*H*;

**int** r = id \* Exec.*H*;

**if**(id==4){

r = Exec.*N*;

}

Vector result = **new** Vector(left.size());

**for** (**int** i = l; i < r; i++) {

result.set(i, 0);

**for** (**int** j = 0; j < left.size(); j++) {

result.set(i, result.get(i) + left.get(j) \* right.get(j, i));

}

}

**return** result;

}

**protected** **static** Matrix add(**final** Matrix left, **final** Matrix right,

**final** **int** id) {

**int** l = (id - 1) \* Exec.*H*;

**int** r = id \* Exec.*H*;

**if**(id==4){

r = Exec.*N*;

}

Matrix result = **new** Matrix(left.size());

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

**for**(**int** j = l; j < r; j++){

result.set(i, j, 0);

result.set(i, j, left.get(i, j) + right.get(i, j));

}

}

**return** result;

}

**protected** **static** Matrix mult(**final** Matrix left, **final** Matrix right,

**final** **int** id) {

**int** l = (id - 1) \* Exec.*H*;

**int** r = id \* Exec.*H*;

**if**(id==4){

r = Exec.*N*;

}

Matrix result = **new** Matrix(left.size());

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

**for** (**int** j = l; j < r; j++) {

result.set(i, j, 0);

**for** (**int** y = 0; y < left.size(); y++) {

result.set(i, j, result.get(i, j) + left.get(i, y)

\* right.get(y, j));

}

}

}

**return** result;

}

**protected** **static** **void** calculation(**final** Vector B, **final** Vector C, **final** Matrix MR, **int** id) {

Vector A1 = Operations.*add*(Operations.*mult*(B, Exec.*MO*, id), Operations.*mult*(C, Operations.*mult*(Exec.*MX*, MR, id), id), id);

**if**(id==4){

**for** (**int** i = (id - 1) \* Exec.*H*; i < Exec.*N*; i++) {

Exec.*A*.set(i, A1.get(i));

}

}**else**{

**for** (**int** i = (id - 1) \* Exec.*H*; i < id \* Exec.*H*; i++) {

Exec.*A*.set(i, A1.get(i));

}

}

}

}

Приложение З. Листинг программы для ПКС ЛП

//PRO course work MPI

//Kornienko Dmytro IO-01

//A = B\*MO+C\*(MR\*MX)

//26.03.2012

// Coursework2.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include "mpi.h"

#include "Data.h"

#include <iostream>

using namespace std;

int N=800;

int P=6;

int H=N/P;

void T0(); void T1(); void T2(); void T3();

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

{

cout<<"count"<<endl;

MPI\_Init(&argc, &argv);

int Tid;

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

if (Tid<4)

{

MPI\_Finalize();

return 0;

}

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

cout<<Tid<<endl;

cout<<"Thread "<<Tid<<" started"<<endl;

double time = MPI\_Wtime();

switch (Tid)

{

case 0:

T0();

time-=MPI\_Wtime();

cout<<"Time: "<<-time<<endl;

break;

case 1:

T1();

break;

case 2:

T2();

break;

case 3:

T3();

break;

}

cout<<"Thread "<<Tid<<" finished"<<endl;

MPI\_Finalize();

return 0;

}

void T0()

{

int \*A = CreateVector(N);

int \*B = CreateVector(N);

int \*C = CreateVector(N);

int \*MRH = CreateMatrix(N, N);

int \*MO = CreateMatrix(N, H);

int \*MX = CreateMatrix(N, H);

MPI\_Status status;

FillVector(B, N, 1);

FillVector(C, N, 1);

MPI\_Send(B+H, N, MPI\_INT, 1, 1, MPI\_COMM\_WORLD);

MPI\_Send(C+H, N, MPI\_INT, 1, 1, MPI\_COMM\_WORLD);

MPI\_Recv(MX, N\*H, MPI\_INT, 1, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(MO, N\*N, MPI\_INT, 1, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(MRH, N\*N, MPI\_INT, 1, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

int t, tt;

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

{

t = tt = 0;

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

{

t += B[j] \* MOh[k\*H+j];

}

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

{

tt += C[k];

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

{

tt \*= MR[i\*N+k] \* MX[k\*H+j];

}

}

A[j] = t + tt;

}

MPI\_Recv(A2h+H, N-3\*H, MPI\_INT, 5, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

OutVector(A, N);

}

void T1()

{

int \*A3h = CreateVector(N-H);

int \*B3H = CreateVector(N);

int \*C3H = CreateVector(N);

int \*MR2H = CreateMatrix(N, N-H);

int \*MO = CreateMatrix(N, H);

int \*MX = CreateMatrix(N, H);

MPI\_Status status;

FillMatrix(MO, N, N, 1);

FillMatrix(MX, N, N, 1);

MPI\_Recv(B3h, N, MPI\_INT, 0, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(C3h, N, MPI\_INT, 0, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(MR2h, N\*(N-H), MPI\_INT, 0, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Send(B2h, N, MPI\_INT, 2, 2, MPI\_COMM\_WORLD);

MPI\_Send(C2h, N, MPI\_INT, 2, 2, MPI\_COMM\_WORLD);

MPI\_Send(MO, N\*H, MPI\_INT, 0, 2, MPI\_COMM\_WORLD);

MPI\_Send(MX, N\*H, MPI\_INT, 0, 2, MPI\_COMM\_WORLD);

MPI\_Send(MRh, N\*(N-2\*H), MPI\_INT, 2, 2, MPI\_COMM\_WORLD);

int t, tt;

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

{

t = tt = 0;

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

{

t += B[j] \* MOh[k\*H+j];

}

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

{

tt += C[k];

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

{

tt \*= MR[i\*N+k] \* MX[k\*H+j];

}

}

A3h[j] = t + tt;

}

MPI\_Recv(A2h, N-4\*H, MPI\_INT, 5, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Send(A2h+H, N-H, MPI\_INT, 3, 5, MPI\_COMM\_WORLD);

}

void T2()

{

int \*A2h = CreateVector(N-2\*H);

int \*B2h = CreateVector(N);

int \*C2h = CreateVector(N);

int \*MR+H = CreateMatrix(N, N-2\*H);

int \*MO = CreateMatrix(N, N);

int \*MX = CreateMatrix(N, 3\*H);

MPI\_Status status;

FillMatrix(MR, N, N, 1);

MPI\_Recv(B2h, N, MPI\_INT, 1, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(C2h, N, MPI\_INT, 1, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(MO, N\*N, MPI\_INT, 3, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(MX, N\*N, MPI\_INT, 3, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Send(MR2h, N\*2\*H, MPI\_INT, 1, 3, MPI\_COMM\_WORLD);

MPI\_Send(Bh, N, MPI\_INT, 3, 3, MPI\_COMM\_WORLD);

MPI\_Send(Ch, N, MPI\_INT, 3, 3, MPI\_COMM\_WORLD);

MPI\_Send(MO, N\*2\*H, MPI\_INT, 1, 3, MPI\_COMM\_WORLD);

MPI\_Send(MX, N\*2\*H, MPI\_INT, 1, 3, MPI\_COMM\_WORLD);

MPI\_Send(MRh, N\*(N-3\*H), MPI\_INT, 3, 3, MPI\_COMM\_WORLD);

int t, tt;

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

{

t = tt = 0;

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

{

t += B[j] \* MOh[k\*H+j];

}

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

{

tt += C[k];

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

{

tt \*= MR[i\*N+k] \* MX[k\*H+j];

}

}

A2h[j] = t + tt;

}

MPI\_Recv(A+H, N-H, MPI\_INT, 1, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Send(A2h, N-2\*H, MPI\_INT, 3, 5, MPI\_COMM\_WORLD);

}

void T3()

{

int \*Ah = CreateVector(N-3\*H);

int \*B = CreateVector(N);

int \*C = CreateVector(N);

int \*MR3h = CreateMatrix(N, N-3\*H);

int \*MO3h = CreateMatrix(N, N-3\*H);

int \*MX4h = CreateMatrix(N, 4\*H);

MPI\_Status status;

MPI\_Recv(Bh, N, MPI\_INT, 2, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(Ch, N, MPI\_INT, 2, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(MO, N\*N, MPI\_INT, 4, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(MX, N\*N, MPI\_INT, 4, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

MPI\_Recv(MRh, N\*H, MPI\_INT, 4, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

int t, tt;

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

{

t = tt = 0;

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

{

t += B[j] \* MOh[k\*H+j];

}

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

{

tt += C[k];

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

{

tt \*= MR[i\*N+k] \* MX[k\*H+j];

}

}

Ah[j] = t + tt;

}

MPI\_Send(Ah, N-3\*H, MPI\_INT, 4, 6, MPI\_COMM\_WORLD);

}

#include "StdAfx.h"

#include <iostream>

#include "Data.h"

using namespace std;

int\* CreateVector(int n)

{

int\* Vector = new int[n];

return Vector;

}

int\* CreateMatrix(int n, int m)

{

int\* Matrix = new int[n\*m];

return Matrix;

}

void FillVector(int\* vector, int n, int c)

{

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

vector[i]=c;

}

void FillMatrix(int\* matrix, int n, int m, int c)

{

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

{

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

{

matrix[i\*m+j]=c;

}

}

}

void OutVector(int\* vector, int n)

{

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

cout<<vector[i]<<'\t';

}

void OutMatrix(int\* matrix, int n, int m)

{

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

{

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

{

cout<<matrix[i\*m+j]<<'\t';

}

cout<<endl;

}

}

#pragma once

int\* CreateVector(int n);

int\* CreateMatrix(int n, int m);

void FillVector(int\* vector, int n, int c);

void FillMatrix(int\* matrix, int n, int m, int c);

void OutVector(int\* vector, int n);

void OutMatrix(int\* matrix, int n, int m);