НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

І.В.Стеценко

МЕТОДИЧНІ ВКАЗІВКИ ДО КОМП'ЮТЕРНОГО ПРАКТИКУМУ З ДИСЦИПЛІНИ «ТЕХНОЛОГІЇ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ»

для студентів факультету інформатики та обчислювальної техніки
спеціальності 121 «Інженерія програмного забезпечення»

Рекомендовано Вченою радою
факультету інформатики та обчислювальної техніки
НТУУ «КПІ ім. І. Сікорського»

Київ - 2021

Стеценко І.В. Методичні вказівки до комп’ютерного практикуму з дисципліни «Технології паралельних обчислень» – К.: НТУУ «КПІ ім. І. Сікорського», 2021. – 19 с.

Рекомендовано Вченою радою ФІОТ НТУУ «КПІ ім. І. Сікорського»
Протокол № ___від « ___» травня 2021 року

Навчально-методичне видання

Методичні вказівки
до комп’ютерного практикуму з дисципліни
«Технології паралельних обчислень»
для студентів факультету інформатики та обчислювальної техніки
спеціальності 121 Інженерія програмного забезпечення

Відповідальний редактор:

Рецензент:

ЗМІСТ

Вступ

Програму навчальної дисципліни «Технології паралельних обчислень» складено відповідно до освітньо-професійної програми підготовки бакалавра напряму 121 «Інженерія програмного забезпечення» професійного спрямування «Інженерія програмного забезпечення комп'ютеризованих систем». Навчальна дисципліна належить до циклу дисциплін професійної та практичної підготовки студента.

Предмет навчальної дисципліни – методи розробки паралельних програм для багатоядерних та багатопроцесорних комп'ютерних систем. Вивчення дисципліни спирається на знання, отримані студентами при вивченні дисциплін «Програмування», «Теорія алгоритмів», «Операційні системи», «Інтелектуальний аналіз даних». Знання та навички, набуті студентом при вивченні дисципліни, використовуються в розробці дипломних проектів бакалавра, а також при вивченні дисциплін підготовки магістрів спеціалізації «Інженерія програмного забезпечення комп'ютеризованих систем».

На лабораторні заняття за навчальним планом спеціальності відведено 36 аудиторних годин та стільки ж самостійної роботи студента. Лабораторні заняття проводяться у вигляді комп'ютерного практикуму.

1 Мета та завдання комп'ютерного практикуму

Під час виконання комп'ютерного практикуму студенти опрацьовують теоретичні знання, що містить лекційний матеріал дисципліни, та набувають практичні навички розробки паралельних програм в багатоядерних та багатопроцесорних обчислювальних системах. Для виконання комп'ютерного практикуму використовуються знання мови програмування Java та C. Окремі завдання комп'ютерного практикуму студенти можуть виконувати іншими мовами паралельного програмування, наприклад, Go.

Основні завдання циклу комп'ютерних практикумів:

- оволодіти навичками розробки паралельних програм з використанням багатопоточної технології;
- оволодіти навичками розробки паралельних програм з використанням розподілених обчислювальних ресурсів на основі технології MPI.

Зміст комп'ютерого практикуму наведений у таблиці 1.1.

Таблиця 1.1

Завдання комп'ютерного практикуму

№ з/п	Назва комп'ютерного практикуму	Номер тижня, отримання завдання	Номер тижня, відведеного для захисту завдання
1.	Створення та запуск потоків: імітація руху більярдних кульок. Найпростіші засоби управління потоками.	1	2

№ з/п	Назва комп'ютерного практикуму	Номер тижня, отримання завдання	Номер тижня, відведеного для захисту завдання
2.	Розробка паралельних алгоритмів множення матриць та дослідження їх ефективності.	1	3
3.	Розробка паралельних програм з використанням механізмів синхронізації: синхронізовані методи, локери, спеціальні типи.	3	5
4.	Розробка паралельних програм з використанням пулів потоків, екзекьюторів та ForkJoinFramework.	5	7
5.	Застосування високорівневих засобів паралельного програмування для побудови алгоритмів імітації та дослідження їх ефективності.	7	9
6.	Розробка паралельного алгоритму множення матриць з використанням MPI-методів обміну повідомленнями «один-до-одного» та дослідження його ефективності.	9	11
7.	Розробка паралельного алгоритму множення матриць з використанням MPI-методів колективного обміну повідомленнями («один-до-багатьох», «багато-до-одного», «багато-до-багатьох») та дослідження його ефективності.	11	13
8.	Розробка паралельних алгоритмів для розподілених систем клієнт-серверної архітектури .	13	15
9.	Розробка паралельного алгоритму самоорганізації моделей дослідження його ефективності	15	17

2 Виконання завдання

Студент має ознайомитись з теоретичним матеріалом у відповідності до теми завдання, використовуючи лекційний матеріал та рекомендовану літературу. Завдання 1-5 виконуються мовою програмування Java (не нижче версії 8.0), 6-9 – MPJ або FastMPJ. Кожне завдання складається з окремих частин, складність виконання яких оцінена відповідною кількістю балів. Студент має право виконати не всі частини завдання. При оцінюванні викладач виставляє оцінку за кожну частину окремо, а сума отриманих балів є оцінкою за завдання в цілому.

При виконанні завдання студент розробляє програмний код та виконує експериментальні дослідження у відповідності до поставленого завдання. Високу оцінку за виконання завдання отримує студент, який 1) представив якісне та самостійне написання коду, ґрунтовне проведення експериментальних досліджень та якісне представлення їх результатів, 2) вчасно захистив виконану роботу. Вчасним вважається завдання, яке здане протягом одного тижня з моменту отримання завдання.

3 Оформлення звіту

Звіт оформлюється у форматі А4, шрифт Times New Roman, 14 пт, міжрядковий інтервал 1, поля 2 см з усіх боків. На титульному аркуші має бути вказана дисципліна, ПІБ студента, група, ПІБ та посада викладача, рядок для виставлення оцінки та дата виставлення оцінки. Зміст звіту для кожного завдання містить 1) текст завдання, 2) лістинг програмного коду, 3) скріншот запуску програми, 4) інші результати, які ілюструють виконання завдання, 5) висновки про переваги та недоліки методів паралельних обчислень, розвитку навичок з яких присвячено завдання комп'ютерного практикуму.

4 Оцінювання завдань комп'ютерного практикуму

За результатами захисту виконаних завдань комп'ютерного практикуму оцінюються практичні навички студента, набути під час вивчення дисципліни. Кожне завдання поділене на частини з відповідною кількістю балів. Якісне та вчасне виконання усіх частин завдання комп'ютерного практикуму оцінюється у 100 балів. На виконання та захист кожного завдання комп'ютерного практикуму студенту дається 1 тиждень. Якщо в цей термін завдання студентом не захищено, він може захистити його протягом наступного тижня, проте його оцінка зменшується на 10%. Якщо протягом двох тижнів завдання студентом не захищене, воно оцінюється у 0 балів. Сумарна оцінка за завдання комп'ютерного практикуму визначається за формулою:

$$\mathrm{P} = \frac{1}{8}\sum D_i,$$

де D_i – 100-бальна оцінка за i-тий комп'ютерний практикум.

Зауваження: необхідною умовою допуску до заліку є кількість балів, набраних за виконання завдань комп'ютерного практикуму, не менша за 60 (із 100 можливих). Тобто, якщо студент не набрав 60 балів за виконання завдань комп'ютерного практикуму ($35 \le \mathrm{P} < 60$), він отримує підсумкову оцінку «Незадовільно». Якщо студент за виконання завдань комп'ютерного практикуму протягом семестру отримав менше 35 балів, він отримує підсумкову оцінку «Не допущено».

Сума балів, набраних студентом протягом семестру, складається з сумарної оцінки за комп'ютерний практикум за формулою за умови, що набрана кількість балів за виконання завдання комп'ютерного практикуму не менша за 60 (із 100 можливих):

$$Z = 0{,}5\cdot\mathrm{P}+0{,}5\cdot\mathrm{T}, \text{ якщо } \mathrm{P} \ge 60,$$
$$Z = \mathrm{P}, \text{ якщо } \mathrm{P} < 60,$$

де Р – оцінка практичних навичок студента, Т – оцінка його теоретичних знань.

5 Завдання до комп'ютерного практикуму

5.1 Завдання до комп'ютерного практикуму 1 «Розробка потоків та дослідження пріоритету запуску потоків»

1. Реалізуйте програму імітації руху більярдних кульок, в якій рух кожної кульки відтворюється в окремому потоці (див. презентацію «Створення та запуск потоків в java» та приклад). Спостерігайте роботу програми при збільшенні кількості кульок. Поясніть результати спостереження. Опишіть переваги потокової архітектури програм. **10 балів.**
2. Модифікуйте програму так, щоб при потраплянні в «лузу» кульки зникали, а відповідний потік завершував свою роботу. Кількість кульок, яка потрапила в «лузу», має динамічно відображатись у текстовому полі інтерфейсу програми. **10 балів.**
3. Виконайте дослідження параметру priority потоку. Для цього модифікуйте програму «Більярдна кулька» так, щоб кульки червоного кольору створювались з вищим пріоритетом потоку, в якому вони виконують рух, ніж кульки синього кольору. Спостерігайте рух червоних та синіх кульок при збільшенні загальної кількості кульок. Проведіть такий експеримент. Створіть багато кульок синього кольору (з низьким пріоритетом) і одну червоного кольору, які починають рух в одному й тому ж самому місці більярдного стола, в одному й тому ж самому напрямку та з однаковою швидкістю. Спостерігайте рух кульки з більшим пріоритетом. Повторіть експеримент кілька разів, значно збільшуючи кожного разу кількість кульок синього кольору. Зробіть висновки про вплив пріоритету потоку на його роботу в залежності від загальної кількості потоків. **20 балів.**
4. Побудуйте ілюстрацію методу join() класу Thread через взаємодію потоків, що відтворюють рух більярдних кульок різного кольору. Поясніть результат, який спостерігається. **10 балів.**
5. Створіть два потоки, один з яких виводить на консоль символ '-', а інший – символ '|'. Запустіть потоки в основній програмі так, щоб вони виводили свої символи в рядок. Виведіть на консоль 100 таких рядків. Поясніть виведений результат. **10 балів.** Використовуючи найпростіші методи управління потоками, добийтесь почергового виведення на консоль символів. **15 балів.**
6. Створіть клас Counter з методами increment() та decrement(), які збільшують та зменшують значення лічильника відповідно. Створіть два потоки, один з яких збільшує 100000 разів значення лічильника, а інший – зменшує 100000 разів значення лічильника. Запустіть потоки на одночасне виконання. Спостерігайте останнє значення лічильника. Поясніть результат. **10 балів.** Використовуючи синхронізований доступ, добийтесь правильної роботи лічильника при одночасній роботі з ним двох і більше потоків. Опрацюйте використання таких способів синхронізації:

синхронізований метод, синхронізований блок, блокування об'єкта. Порівняйте способи синхронізації. **15 балів.**

Приклад програми імітації руху більярдних кульок

```
//
// By Cay S. Horstman
//
```

Лістинг класу **Ball**

```
class Ball {
    private Component canvas;
    private static final int XSIZE = 20;
    private static final int YSIZE = 20;
    private int x = 0;
    private int  y= 0;
    private int dx = 2;
    private int dy = 2;


    public Ball(Component c){
        this.canvas = c;


        if(Math.random()<0.5){
             x = new Random().nextInt(this.canvas.getWidth());
             y = 0;
        }else{
            x = 0;
            y = new Random().nextInt(this.canvas.getHeight());
        }
    }

    public static void f(){
        int a = 0;
    }

    public void draw (Graphics2D g2){
        g2.setColor(Color.darkGray);
        g2.fill(new Ellipse2D.Double(x,y,XSIZE,YSIZE));

    }

    public void move(){
        x+=dx;
        y+=dy;
        if(x<0){
            x = 0;
            dx = -dx;
        }
        if(x+XSIZE>=this.canvas.getWidth()){
            x = this.canvas.getWidth()-XSIZE;
```

```
            dx = -dx;
        }
        if(y<0){
            y=0;
            dy = -dy;
        }
        if(y+YSIZE>=this.canvas.getHeight()){
            y = this.canvas.getHeight()-YSIZE;
            dy = -dy;
        }
        this.canvas.repaint();
    }
}
```

Лістинг класу **BallCanvas**

```
public class BallCanvas extends JPanel{
    private ArrayList<Ball> balls = new ArrayList<>();

    public void add(Ball b){
        this.balls.add(b);
    }
    @Override
    public void paintComponent(Graphics g){
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        for(int i=0; i<balls.size();i++){
            Ball b = balls.get(i);
            b.draw(g2);
        }
    }
}
```

Лістинг класу **BallThread**

```
public class BallThread extends Thread {
    private Ball b;

    public BallThread(Ball ball){
        b = ball;
    }
    @Override
    public void run(){
        try{
            for(int i=1; i<10000; i++){
                b.move();
                System.out.println("Thread name = "
                        + Thread.currentThread().getName());
                Thread.sleep(5);

            }
        } catch(InterruptedException ex){
```

```
        }
    }
}
```

Лістинг класу **BounceFrame**

```
public class BounceFrame extends JFrame {

    private BallCanvas canvas;
    public static final int WIDTH = 450;
    public static final int HEIGHT = 350;

    public BounceFrame() {
        this.setSize(WIDTH, HEIGHT);
        this.setTitle("Bounce programm");

        this.canvas = new BallCanvas();
        System.out.println("In Frame Thread name = "
             + Thread.currentThread().getName());
        Container content = this.getContentPane();
        content.add(this.canvas, BorderLayout.CENTER);

        JPanel buttonPanel = new JPanel();
        buttonPanel.setBackground(Color.lightGray);

        JButton buttonStart = new JButton("Start");
        JButton buttonStop = new JButton("Stop");

        buttonStart.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {

                Ball b = new Ball(canvas);
                canvas.add(b);

                BallThread thread = new BallThread(b);
                thread.start();
                System.out.println("Thread    name    =    "    +
thread.getName());
            }
        });

        buttonStop.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                System.exit(0);
            }

        });
```

```
        buttonPanel.add(buttonStart);
        buttonPanel.add(buttonStop);

        content.add(buttonPanel, BorderLayout.SOUTH);
    }
}
```

Лістинг класу **Bounce**

```
public class Bounce {

    public static void main(String[] args) {
       BounceFrame frame = new BounceFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setVisible(true);
        System.out.println("Thread name = " +
Thread.currentThread().getName());

    }
}
```

5.2 Завдання до комп'ютерного практикуму 2 «Розробка паралельних алгоритмів множення матриць та дослідження їх ефективності»

1. Реалізуйте стрічковий алгоритм множення матриць. Результат множення записуйте в об'єкт класу Result. **30 балів.**
2. Реалізуйте алгоритм Фокса множення матриць. **30 балів.**
3. Виконайте експерименти, варіюючи розмірність матриць, які перемножуються, для обох алгоритмів, та реєструючи час виконання алгоритму. Порівняйте результати дослідження ефективності обох алгоритмів. **20 балів.**
4. Виконайте експерименти, варіюючи кількість потоків, що використовується для паралельного множення матриць, та реєструючи час виконання. Порівняйте результати дослідження ефективності обох алгоритмів. **20 балів.**

5.3 Завдання до комп'ютерного практикуму 3 «Розробка паралельних програм з використанням механізмів синхронізації: синхронізовані методи, локери, спеціальні типи»

1. Реалізуйте програмний код, даний у лістингу, та протестуйте його при різних значеннях параметрів. Модифікуйте програму, використовуючи методи управління потоками, так, щоб її робота була завжди коректною. Запропонуйте три різних варіанти управління. **30 балів**.
2. Реалізуйте приклад Producer-Consumer application (див. https://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html).

Модифікуйте масив даних цієї програми, які читаються, у масив чисел заданого розміру (100, 1000 або 5000) та протестуйте програму. Зробіть висновок про правильність роботи програми. **20 балів**.

3. Реалізуйте роботу електронного журналу групи, в якому зберігаються оцінки з однієї дисципліни трьох груп студентів. Кожного тижня лектор і його 3 асистенти виставляють оцінки з дисципліни за 100-бальною шкалою. **40 балів**.
4. Зробіть висновки про використання методів управління потоками в java. **10 балів**.

Лістинг

```
/**
author Cay Horstmann
*/
public class AsynchBankTest {
   public static final int NACCOUNTS = 10;
   public static final int INITIAL_BALANCE = 10000;

   public static void main(String[] args) {
        Bank b = new Bank(NACCOUNTS, INITIAL_BALANCE);
        int i;
        for (i = 0; i < NACCOUNTS; i++){
             TransferThread t = new TransferThread(b, i,
             INITIAL_BALANCE);
             t.setPriority(Thread.NORM_PRIORITY + i % 2);
             t.start () ;
        }
   }
}

class Bank {

   public static final int NTEST = 10000;
   private final int[] accounts;
   private long ntransacts = 0;

   public Bank(int n, int initialBalance){
        accounts = new int[n];
        int i;
        for (i = 0; i < accounts.length; i++)
             accounts[i] = initialBalance;
        ntransacts = 0;
   }

   public void transfer(int from, int to, int amount) {
        accounts[from] -= amount;
        accounts[to] += amount;
        ntransacts++;
        if (ntransacts % NTEST == 0)
             test();
   }
```

```
   public void test(){
        int sum = 0;
        for (int i = 0; i < accounts.length; i++)
             sum += accounts[i] ;
        System.out.println("Transactions:" + ntransacts
                                              + " Sum: " + sum);
   }

   public int size(){
        return accounts.length;
   }
}

class TransferThread extends Thread {
   private Bank bank;
   private int fromAccount;
   private int maxAmount;
   private static final int REPS = 1000;

   public TransferThread(Bank b, int from, int max){
        bank = b;
        fromAccount = from;
        maxAmount = max;
   }
   @Override
   public void run(){
        while (true) {
          for (int i = 0; i < REPS; i++) {
            int toAccount = (int) (bank.size() * Math.random());
            int amount = (int) (maxAmount * Math.random()/REPS);
            bank.transfer(fromAccount, toAccount, amount);
          }
        }
   }
}
```

5.4 Завдання до комп'ютерного практикуму 4 «Розробка паралельних програм з використанням пулів потоків, екзекьюторів та ForkJoinFramework»

1. Побудуйте алгоритм статистичного аналізу тексту та визначте характеристики випадкової величини «довжина слова в символах» з використанням ForkJoinFramework. **20 балів.** Дослідіть побудований алгоритм аналізу текстових документів на ефективність експериментально. **10 балів.**
2. Реалізуйте один з алгоритмів комп'ютерного практикуму 2 або 3 з використанням ForkJoinFramework та визначте прискорення, яке отримане за рахунок використання ForkJoinFramework. **20 балів.**
3. Розробіть та реалізуйте алгоритм пошуку спільних слів в текстових документах з використанням ForkJoinFramework. **20 балів.**

4. Розробіть та реалізуйте алгоритм пошуку текстових документів, які відповідають заданим ключовим словам (належать до області «Інформаційні технології»), з використанням ForkJoinFramework. **30 балів.**

5.5 Завдання до комп'ютерного практикуму 5 «Застосування високорівневих засобів паралельного програмування для побудови алгоритмів імітації та дослідження їх ефективності»

1. З використанням пулу потоків побудувати алгоритм імітації багатоканальної системи масового обслуговування з обмеженою чергою, відтворюючи функціонування кожного каналу обслуговування в окремій підзадачі. Результатом виконання алгоритму є розраховані значення середньої довжини черги та ймовірності відмови. **40 балів.**
2. З використанням багатопоточної технології організувати паралельне виконання прогонів імітаційної моделі СМО для отримання статистично значимої оцінки середньої довжини черги та ймовірності відмови. **20 балів.**
3. Виводити результати імітаційного моделювання (стан моделі та чисельні значення вихідних змінних) в окремому потоці для динамічного відтворення імітації системи. **20 балів.**
4. Побудувати теоретичні оцінки показників ефективності для одного з алгоритмів практичних завдань 2-5. **20 балів.**

Бонусне завдання

Розробити модель паралельних обчислень для одного з алгоритмів, побудованих при виконанні лабораторних робіт 2-5, з використанням стохастичної мережі Петрі. **25 балів**. Дослідити на моделі зростання часу виконання паралельного алгоритму при збільшенні розміру оброблюваних даних. **25 балів**.

Додаткові вказівки до завдання

Правила обслуговування в багатоканальній СМО з обмеженою чергою (рис. 1):

- Об'єкти надходять на обслуговування в СМО через випадкові інтервали часу з заданим середнім значенням (розподілених, наприклад, за рівномірним законом розподілу).
- Якщо є вільний канал обслуговування, то об'єкт займає його на час обслуговування.
- В противному випадку, об'єкт намагається стати в чергу. Якщо місце в черзі є, то об'єкт займає його. В противному випадку, об'єкт залишає СМО необслугованим (збільшується кількість відмов в обслуговуванні).
- Після завершення обслуговування в каналі об'єкт вважається таким, що завершив обслуговування в СМО (збільшується кількість обслугованих об'єктів), а канал обслуговування - вільним.

- Моделювання здійснюють протягом заданого інтервалу часу (іншим способом є завершення за кількістю об’єктів, які завершили обслуговування). Час моделювання має бути достатньо великим, щоб обслуговування здійснилось для не менш як 1000 об’єктів.
- За результатом імітації підраховують: ймовірність відмови, середня кількість об’єктів в черзі.

Отже, обслуговування об’єктів у багатоканальній СМО схоже на задачу Producer-Consumer, якщо припустити, що потік Producer постачає об’єкти у буфер обмеженої довжини, а потоки Consumer споживають об’єкти з буферу і затримують їх на обслуговування. Затримки, з якими об’єкти надходять та обслуговуються є випадковими величинами. Найчастіше в імітації припускають при надходженні рівномірний закон розподілу, а при обслуговуванні – нормальний закон розподілу.

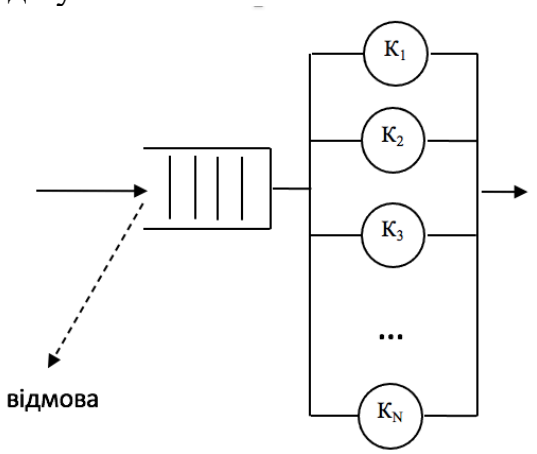

Рис. 1. Схематичне зображення N-канальної СМО з відмовами.

Потік Producer має відкидати об’єкт при невдалій спробі зайняти буфер обмеженої довжини та підраховувати кількість відмов. Потоки Consumer мають підраховувати кількість обслугованих об’єктів. Для підрахунку середньої довжини черги потрібно в окремому потоці через рівні інтервали часу спостерігати значення довжини черги в буфері. Наприкінці імітації з отриманих спостережуваних значень розрахувати середнє значення довжини черги в буфері.

5.6 Завдання до комп’ютерного практикуму 6 «Розробка паралельного алгоритму множення матриць з використанням MPI-методів обміну повідомленнями «один-до-одного» та дослідження його ефективності»

1. Ознайомитись з методами блокуючого та неблокуючого обміну повідомленнями типу point-to-point (див. лекцію та документацію стандарту MPI).
2. Реалізувати алгоритм паралельного множення матриць з використанням розподілених обчислень в MPI з використанням методів блокуючого обміну повідомленнями (лістинг 1). **30 балів.**

3. Реалізувати алгоритм паралельного множення матриць з використанням розподілених обчислень в MPI з використанням методів неблокуючого обміну повідомленнями. **30 балів.**
4. Дослідити ефективність розподіленого обчислення алгоритму множення матриць при збільшенні розміру матриць та при збільшенні кількості вузлів, на яких здійснюється запуск програми. Порівняйте ефективність алгоритму при використанні блокуючих та неблокуючих методів обміну повідомленнями. **40 балів.**

Лістинг програми «MPI Matrix Multiply» (OpenMPI)

```
/*FILE: mpi_mm.c
* By Blaise Barney
*/
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define NRA 62          /* number of rows in matrix A */
#define NCA 15          /* number of columns in matrix A */
#define NCB 7           /* number of columns in matrix B */
#define MASTER 0        /* taskid of first task */
#define FROM_MASTER 1 /* setting a message type */
#define FROM_WORKER 2 /* setting a message type */
int main (int argc, char *argv[]) {
     int numtasks,
     taskid,
     numworkers,
     source,
     dest,
     rows,          /* rows of matrix A sent to each worker */
     averow, extra, offset,
           i, j, k, rc;
     double a[NRA][NCA], /* matrix A to be multiplied */
            b[NCA][NCB], /* matrix B to be multiplied */
            c[NRA][NCB]; /* result matrix C */
     MPI_Status status;
     MPI_Init( &argc, &argv);
     MPI_Comm_size( MPI_COMM_WORLD, &numtasks);
     MPI_Comm_rank( MPI_COMM_WORLD, &taskid);
     if (numtasks < 2 ) {
         printf("Need at least two MPI tasks. Quitting...\n");
         MPI_Abort(MPI_COMM_WORLD, rc);
         exit(1);
     }
     numworkers = numtasks-1;
     if (taskid == MASTER) {
         printf("mpi_mm has started with %d tasks.\n", numtasks);
         for (i=0; i<NRA; i++)
           for (j=0; j<NCA; j++)
                   a[i][j]= 10;
         for (i=0; i<NCA; i++)
```

```
      for (j=0; j<NCB; j++)
               b[i][j]= 10;

    averow = NRA/numworkers;
    extra = NRA%numworkers;
    offset = 0;
    for (dest=1; dest<=numworkers; dest++) {
      rows = (dest <= extra) ? averow+1 : averow;
      printf("Sending %d rows to task %d offset= %d\n",
                                          rows,dest,offset);
      MPI_Send(&offset, 1, MPI_INT, dest, FROM_MASTER,
                                            MPI_COMM_WORLD);
      MPI_Send(&rows, 1, MPI_INT, dest, FROM_MASTER,
                                            MPI_COMM_WORLD);
      MPI_Send(&a[offset][0], rows*NCA, MPI_DOUBLE, dest,
                                    ROM_MASTER,MPI_COMM_WORLD);
      MPI_Send(&b, NCA*NCB, MPI_DOUBLE, dest, FROM_MASTER,
                                             MPI_COMM_WORLD);
      offset = offset + rows;
   }
/* Receive results from worker tasks */
for (source=1; source<=numworkers; source++) {
        MPI_Recv(&offset, 1, MPI_INT, source, FROM_WORKER,
                                    MPI_COMM_WORLD, &status);
         MPI_Recv(&rows, 1, MPI_INT, source, FROM_WORKER,
                                    MPI_COMM_WORLD, &status);
         MPI_Recv(&c[offset][0], rows*NCB, MPI_DOUBLE,
                                      source,FROM_WORKER,
                                     MPI_COMM_WORLD,&status);
        printf("Received results from task %d\n", id);
}
/* Print results */
printf("****\n");
printf("Result Matrix:\n");
for (i=0; i<NRA; i++) {
          printf("\n");
          for (j=0; j<NCB; j++)
                   printf("%6.2f ", c[i][j]);
}
printf("\n********\n");
printf ("Done.\n");
}
/******** worker task *****************/
else{ /* if (taskid > MASTER) */
    MPI_Recv(&offset, 1, MPI_INT, MASTER, FROM_MASTER,
                                    MPI_COMM_WORLD,&status);
    MPI_Recv(&rows, 1, MPI_INT, MASTER, FROM_MASTER,
                                    MPI_COMM_WORLD, &status);
    MPI_Recv(&a, rows*NCA, MPI_DOUBLE, MASTER, FROM_MASTER,
                                    MPI_COMM_WORLD, &status);
    MPI_Recv(&b, NCA*NCB, MPI_DOUBLE, MASTER, FROM_MASTER,
                                    MPI_COMM_WORLD, &status);
    for (k=0; k<NCB; k++)
```

```
                for (i=0; i<rows; i++) {
                    c[i][k] = 0.0;
                    for (j=0; j<NCA; j++)
                              c[i][k] = c[i][k] + a[i][j] * b[j][k];
                }
          MPI_Send(&offset, 1, MPI_INT, MASTER, FROM_WORKER,
                                                  MPI_COMM_WORLD);
               MPI_Send(&rows, 1, MPI_INT, MASTER, FROM_WORKER,
                                                  MPI_COMM_WORLD);
               MPI_Send(&c, rows*NCB, MPI_DOUBLE, MASTER,
                                      FROM_WORKER, MPI_COMM_WORLD);
     }
     MPI_Finalize();
}
```

5.7 Завдання до комп'ютерного практикуму 7 «Розробка паралельного алгоритму множення матриць з використанням MPI-методів колективного обміну повідомленнями («один-до-багатьох», «багато-до-одного», «багато-до-багатьох») та дослідження його ефективності»

1. Ознайомитись з методами колективного обміну повідомленнями типу «один-до-багатьох», «багато-до-одного», «багато-до-багатьох» (див. лекцію та документацію стандарту MPI).
2. Реалізувати алгоритм паралельного множення матриць з використанням розподілених обчислень в MPI з використанням методів колективного обміну повідомленнями. **40 балів.**
3. Дослідити ефективність розподіленого обчислення алгоритму множення матриць при збільшенні розміру матриць та при збільшенні кількості вузлів, на яких здійснюється запуск програми. Порівняйте ефективність алгоритму при використанні методів обміну повідомленнями «один-до-одного», «один-до-багатьох», «багато-до-одного», «багато-до-багатьох». **60 балів.**

5.8 Завдання до комп'ютерного практикуму 8 «Розробка алгоритмів для розподілених систем клієнт-серверної архітектури»

1. Розробити веб-застосування клієнт-серверної архітектури, що реалізує алгоритм множення матриць або інший, який був Вами реалізований в рамках курсу «Технології паралельних обчислень», на стороні сервера з використанням паралельних обчислень. Розгляньте два варіанти реалізації 1) дані для обчислень знаходяться на сервері та 2) дані для обчислень знаходяться на клієнтській частині застосування. **60 балів.**
2. Дослідити швидкість виконання запиту користувача при різних обсягах даних. **30 балів.**
3. Порівняти реалізацію алгоритму в клієнт-серверній системі та в розподіленій системі з рівноправними процесорами. **10 балів.**

5.9 Завдання до комп'ютерного практикуму 9 «Розробка паралельного алгоритму самоорганізації моделей дослідження його ефективності»

1. Для алгоритму самоорганізації моделей (див. https://github.com/StetsenkoInna/GMDH) розробити його паралельну реалізацію (або багатопоточну, або MPI). **40 балів.**
2. Дослідити ефективність розробленого алгоритму при збільшенні кількості опорних функцій та кількості ресурсів (потоків або процесів), на яких запускається програма. Результати дослідження представити графічно. **30 балів.**
3. Для даних, які відповідають реальному об'єкту дослідження (дослідження кліматичних змін, енергоспоживання, економічних показників розвитку регіону та інше), виконати розрахунок моделі оптимальної складності та оцінку її якості з використанням розробленого алгоритму. **20 балів.**
4. Зробити висновки щодо використання паралельних обчислень в розподілених системах для прикладних задач. **10 балів.**

6 Рекомендована література

6.1 Базова

1. The Java Tutorials Lesson:Concurrency [Електронний ресурс] – Режим доступу: https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html
2. FastMPJ User's Guide [Електронний ресурс] – Режим доступу: http://gac.udc.es/~rreye/fastmpj/doc/UsersGuide.pdf
3. Сайт Українського інституту інформаційних технологій в освіті. Дисципліна «Технології розподілених систем та паралельних обчислень» викладача Стеценко Інни Вячеславівни [Електронний ресурс] – Режим доступу: https://do.ipo.kpi.ua/course/view.php?id=4093
4. Стіренко С. Г. Засоби паралельного програмування / С. Г. Стіренко. Д. В. Грибенко. О. І. Зіненко. А. В. Михайленко – Київ. 2012. – 183 с. [Електронний ресурс] – Режим доступу: http://hpcc.kpi.ua/hpc-book/
5. Lea D. Concurrent programming in Java: design principles and patterns / D. Lea – Addison-Wesley Professional. 2000. – 411p.
6. Foster I. Designing and Building Parallel Programs [Електронний ресурс] – Режим доступу: http://www.mcs.anl.gov/~itf/dbpp/text/book.html

6.2 Допоміжна

1. Аксак Н.Г. Паралельні та розподілені обчислення: підруч./ НГ.Аксак. О.Г. Руденко. А.М.Гуржій. – Х.:Компанія СМІТ. 2009. – 480с.