Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Саратовский государственный технический университет

имени Гагарина Ю.А.»

Институт электронной техники и приборостроения

Кафедра Информационная безопасность автоматизированных систем

Направление 10.05.03 Информационная безопасность автоматизированных систем

**Практическая работа №1**

по дисциплине «Параллельные системы и их программирование»

по теме **«**Обработка элементов вектора»

|  |  |
| --- | --- |
|  | Выполнил: студент 4 курса  учебной группы с-ИБС42  очной формы обучения  Солодилов В.В.  Проверил: профессор. каф. ИБС  Кондратов Д.В. |

Саратов 2022

**Задания**

1. Реализуйте последовательную обработку элементов вектора, например, умножение элементов вектора на число. Число элементов вектора задается параметром N.
2. Реализуйте многопоточную обработку элементов вектора, используя разделение вектора на равное число элементов. Число потоков задается параметром M.
3. Выполните анализ эффективности многопоточной обработки при разных параметрах N (10, 100, 1000, 100000) и M (2, 3, 4, 5, 10) Результаты представьте в табличной форме.
4. Выполните анализ эффективности при усложнении обработки каждого элемента вектора.
5. Исследуйте эффективность разделения по диапазону при неравномерной вычислительной сложности обработки элементов вектора.
6. Исследуйте эффективность параллелизма при круговом разделении элементов вектора. Сравните с эффективностью разделения по диапазону.

**Вопросы**

1. Почему эффект от распараллеливания наблюдается только при большем числе элементов?
2. Почему увеличение сложности обработки повышает эффективность многопоточной обработки?
3. Какое число потоков является оптимальным для конкретной вычислительной системы?
4. Почему неравномерность загрузки потоков приводит к снижению эффективности многопоточной обработки?
5. Какие другие варианты декомпозиции позволяют увеличить равномерность загрузки потоков?
6. В какой ситуации круговая декомпозиция не обеспечивает равномерную загрузку потоков?

*Задание 3.* Анализ эффективности многопоточной обработки при разных параметрах N и M.

Выполним анализ эффективности многопоточной обработки при разных параметрах N и M для операции возведения числа в пятую степень:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **M\N** | **10** | **100** | **1000** | **100000** |
| **1** | 44 мкс | 14 мкс | 70 мкс | 4917 мкс |
| **2** | 130 мкс | 125 мкс | 201 мкс | 1013 мкс |
| **3** | 218 мкс | 214 мкс | 195 мкс | 914 мкс |
| **4** | 368 мкс | 305 мкс | 280 мкс | 991 мкс |
| **5** | 476 мкс | 374 мкс | 388 мкс | 998 мкс |
| **10** | 698 мкс | 719 мкс | 721 мкс | 1370 мкс |

Таблица 1. Время выполнения операции при разных параметрах N и M

На основе данных, приведенных в таблице, можно сделать вывод, что при небольшом числе элементов вектора эффективность многопоточной обработки значительно ниже относительно однопоточной обработки той же операции, так как большая часть времени тратится на организацию работы большого числа потоков.

Однако, при большем количестве элементов видно, что эффективность многопоточной обработки возрастает более чем в 5 раз, что делает данный подход более предпочтительным в данной ситуации. Также при большем числе элементов вектора возрастает сама эффективность разбиения выполнения операции на большее число потоков.

*Задание 4.* Анализ эффективности при усложнении обработки каждого элемента вектора.

Для проведения анализа возьмём параметр K = 100 и K = 10000 и сравним полученные результаты с однопоточной обработкой. Основной операцией является возведение числа в пятую степень:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **M\N** | **10** | **100** | **1000** | **100000** |
| **1** | 92 мкс | 425 мкс | 3346 мкс | 180210 мкс |
| **2** | 120 мкс | 133 мкс | 194 мкс | 782 мкс |
| **3** | 237 мкс | 209 мкс | 272 мкс | 1126 мкс |
| **4** | 289 мкс | 270 мкс | 382 мкс | 1250 мкс |
| **5** | 614 мкс | 470 мкс | 384 мкс | 1287 мкс |
| **10** | 742 мкс | 724 мкс | 730 мкс | 1555 мкс |

Таблица 2. Время выполнения операции при K = 100

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **M\N** | **10** | **100** | **1000** | **100000** |
| **1** | 3568 мкс | 19614 мкс | 176935 мкс | 18094997 мкс |
| **2** | 125 мкс | 129 мкс | 223 мкс | 1329 мкс |
| **3** | 215 мкс | 192 мкс | 320 мкс | 2012 мкс |
| **4** | 287 мкс | 322 мкс | 385 мкс | 1161 мкс |
| **5** | 356 мкс | 354 мкс | 511 мкс | 1080 мкс |
| **10** | 653 мкс | 804 мкс | 1096 мкс | 1415 мкс |

Таблица 3. Время выполнения операции при K = 10000

При увеличении параметра K значительно возрастает время выполнения операции при однопоточной обработке, в то время как при многопоточной обработке разница во времени при значительном усложнении обработки каждого элемента вектора практически незаметна.

Помимо этого, при K = 100 эффективность многопоточной обработки уже становится видна при числе элементов N = 100, а при K = 10000 она уже проявляется при числе элементов N = 10. Причём в обоих случаях эффективность многопоточной обработки возрастает на порядок по сравнению с однопоточной.

*Задание 5.* Анализ эффективности разделения по диапазону при неравномерной вычислительной сложности обработки элементов вектора.

Выполним анализ разделения по диапазону при неравномерной вычислительной сложности. Результат отразим в таблице:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **M\N** | **10** | **100** | **1000** | **100000** |
| **1** | 49 мкс | 213 мкс | 11536 мкс | 91290818 мкс |
| **2** | 120 мкс | 131 мкс | 184 мкс | 1017 мкс |
| **3** | 218 мкс | 189 мкс | 245 мкс | 1102 мкс |
| **4** | 277 мкс | 287 мкс | 263 мкс | 1071 мкс |
| **5** | 573 мкс | 392 мкс | 340 мкс | 1228 мкс |
| **10** | 730 мкс | 697 мкс | 745 мкс | 1543 мкс |

Таблица 4. Время выполнения операции при неравномерной сложности обработки

На основе результатов таблицы можно сделать вывод, что при однопоточной обработке время выполнения программы возрастает не так значительно при сравнении с временем при увеличении сложности обработки каждого элемента вектора, однако при N = 100000 время на порядок больше, чем при усложненной обработке каждого элемента массива.

В связи с этим многопоточная обработка является более выгодным вариантом. В данном случае можно видеть, что время выполнения программы при многопоточной обработке даже ниже, чем при усложненной обработке каждого элемента массива при K = 10000. В то же время до N = 100 время выполнения программы остаётся практически на одном уровне и возрастает лишь при N = 100000, но не так значительно, как при однопоточной обработке.

С увеличением числа потоков эффективность выполнения операции возрастает только при N = 100000, т.е. при значительном числе элементов. Во всех остальных случаях наблюдается только рост времени выполнения программы.

*Задание 6.* Анализ эффективности параллелизма при круговом разделении элементов вектора.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **M\N** | **10** | **100** | **1000** | **100000** |
| **1** | 46 мкс | 204 мкс | 11348 мкс | 91592662 мкс |
| **2** | 165 мкс | 165 мкс | 141 мкс | 795 мкс |
| **3** | 203 мкс | 205 мкс | 195 мкс | 889 мкс |
| **4** | 259 мкс | 250 мкс | 255 мкс | 834 мкс |
| **5** | 311 мкс | 295 мкс | 342 мкс | 809 мкс |
| **10** | 590 мкс | 596 мкс | 618 мкс | 1055 мкс |

Таблица 4. Время выполнения операции при неравномерной сложности обработки

Если сравнить полученный результат с результатом разбиения элементов по диапазону, то можно увидеть, что эффективность кругового разбиения возрастает при числе элементов массива N = 100. При дальнейшем увеличении элементов вектора эффективность параллелизма при данном методе повышается.

# **Ответы на вопросы**

*Вопрос 1.* Эффект от распараллеливания наблюдается только при большем числе элементов, так как при малом объеме элементов массива, накладываются расходы, связанные с организацией многопоточной обработки, которые существенно превышают выигрыш от параллельности обработки.

*Вопрос 2.* Увеличение сложности обработки повышает эффективность многопоточной обработки из-за того, что при многопоточной обработке происходит более эффективное использование потоков и кэш-памяти процессора.

*Вопрос 3.* На данной вопрос нельзя ответить однозначно, так как в каждом конкретном случае нужно находить баланс между сложностью выполняемых операций и числом используемых потоков процессора, не забывая про общее время выполнения всех операций.

*Вопрос 4.* Данная ситуация происходит из-за неравномерно распределенной сложности обработки каждого элемента вектора, ведь при росте индекса элемента сам элемент становится обрабатывать значительно сложнее. В результате чего при разделении такого вектора по диапазону происходит ситуация, когда в один поток попадают более «простые» элементы, чем в другой. В связи с этим возникает неравномерность нагрузки каждого потока, что снижает эффективность параллелизма.

*Вопрос 5.* Круговая декомпозиция позволяет увеличить равномерность загрузки потоков.

*Вопрос 6.* Круговая декомпозиция не обеспечивает равномерную загрузку потоков в случае, если определены четное число элементов и нечетное число потоков.

# **Вывод**

В рамках данной практической работы была реализована программа на языке Java, позволяющая производить обработку вектора различными способами: линейно и параллельно с диапазонным или круговым разбиением. При анализе эффективности параллелизма было выявлено, что при небольшом числе элементов наиболее эффективным является однопоточная реализация программы, однако при росте сложности операций и числа элементов многопоточная обработка становится эффективнее, причём в отдельных случаях на порядок. Нельзя не отметить, что при усложнении операций круговое разбиение элементов массива сокращает время выполнения операций даже при небольшом числе элементов по сравнению с разбиением по диапазону. Это достигается путём равномерного распределения нагрузки на потоки процессора.

**Листинг программы**

import java.util.Scanner;

import java.util.concurrent.atomic.AtomicInteger;

public class Main {

static Scanner scanner = new Scanner(System.in);

public static void main(String[] args) {

int K = 0;

System.out.print("""

Выберите задачу:

0 - Базовая обработка

1 - Усложненная обработка

2 - Обработка при неравномерной сложности:\s""");

int taskValue = scanner.nextInt();

if (taskValue < 0 || taskValue > 2) {

while (taskValue < 0 || taskValue > 2) {

System.out.print("Повторите ввод: ");

taskValue = scanner.nextInt();

}

} else if (taskValue == 1) {

System.out.print("Введите K = ");

K = scanner.nextInt();

} else K = 100;

System.out.println("\nПоследовательная обработка:");

for (int i = 10; i <= 100000; i = i \* 10) {

long time1 = System.nanoTime() / 1000;

sequentialFunction(createStartVector(i), 5, K, taskValue);

long time2 = System.nanoTime() / 1000 - time1;

System.out.println("1 поток " + i + " вектор = " + time2 + " мкс");

}

System.out.print("\nПараллельная обработка по диапазону:");

for (int i = 10; i <= 1000000; i = i \* 10) {

long time1 = System.nanoTime() / 1000;

parallelFunction(createStartVector(i), 2, 5, K, taskValue);

long time2 = System.nanoTime() / 1000 - time1;

parallelFunction(createStartVector(i), 3, 5, K, taskValue);

long time3 = System.nanoTime() / 1000 - time1 - time2;

parallelFunction(createStartVector(i), 4, 5, K, taskValue);

long time4 = System.nanoTime() / 1000 - time1 - time2 - time3;

parallelFunction(createStartVector(i), 5, 5, K, taskValue);

long time5 = System.nanoTime() / 1000 - time1 - time2 -time3 - time4;

parallelFunction(createStartVector(i), 10, 5, K, taskValue);

long time6 = System.nanoTime() / 1000 - time1 - time2 -time3 - time4 - time5;

System.out.println("\n2 потока " + i + " вектор = " + time2 + " мкс");

System.out.println("3 потока " + i + " вектор = " + time3 + " мкс");

System.out.println("4 потока " + i + " вектор = " + time4 + " мкс");

System.out.println("5 потоков " + i + " вектор = " + time5 + " мкс");

System.out.println("10 потоков " + i + " вектор = " + time6 + " мкс");

}

System.out.print("\nПараллельная обработка при круговом разделении:");

for (int i = 10; i <= 1000000; i = i \* 10) {

long time1 = System.nanoTime() / 1000;

parallelCircleFunction(createStartVector(i), 2, 5, K, taskValue);

long time2 = System.nanoTime() / 1000 - time1;

parallelCircleFunction(createStartVector(i), 3, 5, K, taskValue);

long time3 = System.nanoTime() / 1000 - time1 - time2;

parallelCircleFunction(createStartVector(i), 4, 5, K, taskValue);

long time4 = System.nanoTime() / 1000 - time1 - time2 - time3;

parallelCircleFunction(createStartVector(i), 5, 5, K, taskValue);

long time5 = System.nanoTime() / 1000 - time1 - time2 -time3 - time4;

parallelCircleFunction(createStartVector(i), 10, 5, K, taskValue);

long time6 = System.nanoTime() / 1000 - time1 - time2 -time3 - time4 - time5;

System.out.println("\n2 потока " + i + " вектор = " + time2 + " мкс");

System.out.println("3 потока " + i + " вектор = " + time3 + " мкс");

System.out.println("4 потока " + i + " вектор = " + time4 + " мкс");

System.out.println("5 потоков " + i + " вектор = " + time5 + " мкс");

System.out.println("10 потоков " + i + " вектор = " + time6 + " мкс");

}

System.exit(0);

}

public static double[] createStartVector(int N) {

double[] startVector = new double[N];

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

startVector[i] = i;

}

return startVector;

}

public static void sequentialFunction(double[] startVector, int C, int K, int taskValue) {

//int K = 10000;

double[] endVector = new double[startVector.length];

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

switch (taskValue) {

case 0 -> endVector[i] = Math.pow(startVector[i], C);

case 1 -> {

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

endVector[i] += Math.pow(startVector[i], C);

}

}

case 2 -> {

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

endVector[i] += Math.pow(startVector[i], C);

}

}

}

}

}

public static void parallelFunction(double[] startVector, int M, int C, int K, int taskValue) {

double[] endVector = new double[startVector.length];

AtomicInteger tempValue = new AtomicInteger();

Object lock = new Object();

Runnable runnable = () -> {

synchronized (lock) {

float temp = (float) endVector.length / M;

for (int i = tempValue.get(); i < tempValue.get() + temp; i++) {

if (i >= endVector.length) break;

else {

switch (taskValue) {

case 0 -> endVector[i] = Math.pow(startVector[i], C);

case 1 -> {

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

endVector[i] += Math.pow(startVector[i], C);

}

}

case 2 -> {

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

endVector[i] += Math.pow(startVector[i], C);

}

}

}

}

}

tempValue.set(tempValue.get() + (int) Math.ceil(temp));

}

};

Thread[] threads = new Thread[M];

synchronized (lock) {

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

threads[i] = new Thread(runnable);

threads[i].start();

}

}

}

public static void parallelCircleFunction(double[] startVector, int M, int C, int K, int taskValue) {

AtomicInteger tempValue1 = new AtomicInteger();

AtomicInteger tempValue2 = new AtomicInteger();

Object lock = new Object();

double[] endVector = new double[startVector.length];

tempValue2.set(1);

Runnable runnable1 = () -> {

synchronized (lock) {

int j = 0;

int k = 0;

for (int i = tempValue1.get(); i < endVector.length; i++) {

if (i % 2 == 0) {

switch (taskValue) {

case 0 -> {

endVector[i] = Math.pow(startVector[i], C);

j++;

}

case 1 -> {

for (int b = 0; b < K; b++) {

endVector[i] += Math.pow(startVector[i], C);

}

j++;

}

case 2 -> {

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

endVector[i] += Math.pow(startVector[i], C);

}

j++;

}

}

}

k = i;

if (j == endVector.length / M) break;

}

tempValue1.set(tempValue1.get() + k + 2);

}

};

Runnable runnable2 = () -> {

synchronized (lock) {

int j = 0;

int k = 0;

for (int i = tempValue2.get(); i < endVector.length; i++) {

if (i % 2 != 0) {

switch (taskValue) {

case 0 -> {

endVector[i] = Math.pow(startVector[i], C);

j++;

}

case 1 -> {

for (int b = 0; b < K; b++) {

endVector[i] += Math.pow(startVector[i], C);

}

j++;

}

case 2 -> {

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

endVector[i] += Math.pow(startVector[i], C);

}

j++;

}

}

}

k = i;

if (j == endVector.length / M) break;

}

tempValue2.set(tempValue2.get() + k + 2);

}

};

Thread[] threads = new Thread[M];

synchronized (lock) {

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

if (i % 2 == 0) {

threads[i] = new Thread(runnable1);

threads[i].start();

} else {

threads[i] = new Thread(runnable2);

threads[i].start();

}

}

}

}

}