НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №6

з дисципліни **«**Програмування паралельних комп’ютерних систем**»**

**«Java. Монітори»**

Виконав:

студент 3 курсу

ФІОТ гр. ІО-34

Кривоносов Олексій

Перевірив:

Корочкін О. В.

Київ – 2016 р.

Тема: Програмування для комп’ютерних систем зі спільною пам’яттю.

Розробити програму для розв’язання в ПКС зі СП математичної задачі: MA = (B∙C)∙MO + α∙(MT∙MR)

Мова програмування: Java

Засоби організації взаємодії: монітори



Структурна схема ПКС

**Виконання роботи:**

**Етап 1. Побудова паралельного алгоритму**

1. vi = BH\*CH, *i* =
2. v = v + vi, *i* =
3. MAH = v∙MOH + α∙(MTH∙MR)

Спільний ресурс: α, v, MR

**Етап 2. Розроблення алгоритмів роботи кожного процесу**

Зручним варіантом реалізації є написання єдиного алгоритму для всіх задач. Кожна задача має свій ідентифікатор tid (tid=).

|  |  |
| --- | --- |
| **Задача Тi (i=)** | КД |
| 1. Якщо tid = 1, Введення MT,MR |  |
| 2. Якщо tid = 1, **Сигнал** про завершення вводу. |  |
| 3. Якщо tid = 3, Введення B |  |
| 4. Якщо tid = 2, **Сигнал** про завершення вводу. |  |
| 5. Якщо tid = 4, Введення C, α, MO |  |
| 6. Якщо tid = 4, **Сигнал** про завершення вводу. |  |
| 7. **Чекати** завершення вводу в інших задачах |  |
| 8. Обчислення 1: vi = BH∙CH |  |
| 9. Обчислення 2: v = v + vi | КД |
| 10. **Сигнал** про завершення обчислення 2. |  |
| 11. **Чекати** завершення обчислення 2 в інших задачах |  |
| 12. Копіювати αi := α, vi = v, MRi = MR | КД |
| 13. Обчислення 3: MAH = vi∙MOH + αi∙(MTH∙MRi) |  |
| 14. Якщо tid !=4, **Сигнал** про завершення обчислення 3 |  |
| 15. Якщо tid = 6, **Чекати** на завершення обчислення 3 в інших задачах |  |
| 16. Якщо tid = 6, Вивести MA |  |

**Етап 3. Розроблення структурної схеми взаємодії задач**



Етап 4. Розроблення програми

/\*

-- Parallel and Distributed Computing

-- Laboratory work #6 Java

--

-- MA = (B\*C)\*MO + alpha\*(MT\*MR)

--

-- Author: Kryvonosov Oleksii, group IO-34

-- Date: 22.04.2016

------------------------------------------------------------------ \*/

package PRO6;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

**public class Executor** {

public static int N = 6;

public static int P = 6;

public static int H = N / P;

public static Matrix MT = new Matrix(N);

public static Matrix MA = new Matrix(N);

public static Matrix MO = new Matrix(N);

public static Vector B = new Vector(N);

public static Vector C = new Vector(N);

public static void main(String[] args) {

System.out.println("Lab 6 started");

ExecutorService service = Executors.newFixedThreadPool(P);

GeneralResourseMonitor resourseMonitor = new GeneralResourseMonitor();

SynchronizationMonitor syncMonitor = new SynchronizationMonitor();

for (int id = 1; id <= P; id++) {

service.execute(new CalculationWorker(resourseMonitor, syncMonitor, id));

}

service.shutdown();

}

}

package PRO6;

import java.util.ArrayList;

import java.util.List;

**public class CalculationWorker implements Runnable** {

private int tid;

private GeneralResourseMonitor resMonitor;

private SynchronizationMonitor syncMonitor;

public CalculationWorker(GeneralResourseMonitor resourseMonitor,

SynchronizationMonitor syncMonitor, int id) {

this.resMonitor = resourseMonitor;

this.syncMonitor = syncMonitor;

this.tid = id;

}

@Override

public void run() {

System.out.println("Thread "+tid + " started.");

switch (this.tid) {

case 1:

Executor.MT = CalculateUtils.inputMatrix(1);

Matrix MR = CalculateUtils.inputMatrix(1);

resMonitor.setMR(MR);

syncMonitor.inputSignal();

break;

case 3:

Executor.B = CalculateUtils.inputVector(1);

syncMonitor.inputSignal();

break;

case 4:

Executor.C = CalculateUtils.inputVector(1);

Executor.MO = CalculateUtils.inputMatrix(1);

int alpha = 1;

resMonitor.setAlpha(alpha);

syncMonitor.inputSignal();

break;

}

syncMonitor.waitForInput();

if (tid == 1) {

List<ForkSum> tasks = new ArrayList<ForkSum>();

for (int i = 0; i < Executor.P; i++) {

tasks.add(new ForkSum(i \* Executor.H, (i + 1) \* Executor.H, Executor.B, Executor.C, resMonitor));

}

for (ForkSum forkSum: tasks){

forkSum.fork();

forkSum.join();

}

}

syncMonitor.calcVSignal();

syncMonitor.waitForCalcV();

int alphai = resMonitor.copyAlpha();

int vi = resMonitor.copyV();

Matrix MR = resMonitor.copyMR();

CalculateUtils.operations(vi, alphai, MR, tid);

if (tid == 3){

syncMonitor.waitForCalcMA();

CalculateUtils.outputMatrix(Executor.MA);

}else{

syncMonitor.calcMASignal();

}

System.out.println("Thread "+tid + " finished.");

}

}

package PRO6;

import java.util.concurrent.RecursiveAction;

**public class ForkSum extends RecursiveAction** {

private int start, end;

private Vector left, right;

GeneralResourseMonitor generalResourseMonitor;

ForkSum(int s, int e, Vector l, Vector r, GeneralResourseMonitor generalResourseMonitor) {

this.start = s;

this.end = e;

this.left = l;

this.right = r;

this.generalResourseMonitor = generalResourseMonitor;

}

@Override

protected void compute() {

int vi = scalVec(left, right);

generalResourseMonitor.addV(vi);

}

private int scalVec(Vector left, Vector right) {

int result = 0;

for (int i=start; i<end; i++) {

result += left.get(i) \* right.get(i);

}

return result;

}

}

package PRO6;

**public class GeneralResourseMonitor** {

private Matrix MR;

private int alpha;

private int v;

private int e;

public synchronized void setAlpha(int value) {

this.alpha = value;

}

public synchronized void addV(int value) {

this.v += value;

}

public synchronized void setMR(Matrix value) {

this.MR = value;

}

public synchronized int copyAlpha() {

return this.alpha;

}

public synchronized int copyV() {

return this.v;

}

public synchronized Matrix copyMR() {

return this.MR;

}

}

package PRO6;

**public class SynchronizationMonitor** {

private int inputFlag;

private int vFlag;

private int eFlag;

private int AFlag;

public synchronized void inputSignal() {

inputFlag++;

if (inputFlag >= 3)

notifyAll();

}

public synchronized void calcVSignal() {

vFlag++;

if (vFlag >= Executor.P)

notifyAll();

}

public synchronized void calcMASignal() {

AFlag++;

if (AFlag == Executor.P - 1)

notifyAll();

}

public synchronized void waitForInput() {

try {

if (inputFlag < 3) wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public synchronized void waitForCalcV() {

try {

if (vFlag < Executor.P)

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public synchronized void waitForCalcMA() {

try {

if (AFlag < Executor.P - 1)

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

package PRO6;

public class SynchronizationMonitor {

private int inputFlag;

private int vFlag;

private int eFlag;

private int AFlag;

public synchronized void inputSignal() {

inputFlag++;

if (inputFlag >= 3)

notifyAll();

}

public synchronized void calcVSignal() {

vFlag++;

if (vFlag >= Executor.P)

notifyAll();

}

public synchronized void calcMASignal() {

AFlag++;

if (AFlag == Executor.P - 1)

notifyAll();

}

public synchronized void waitForInput() {

try {

if (inputFlag < 3) wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public synchronized void waitForCalcV() {

try {

if (vFlag < Executor.P)

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public synchronized void waitForCalcMA() {

try {

if (AFlag < Executor.P - 1)

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}