Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 3 з дисципліни
«Паралельні та розподілені обчислення»

«Використання виконувачів із пакету *java.util.concurrent*»

Варіант 7

Виконав студент ІП-15, Буяло Дмитро Олександрович
(шифр, прізвище, ім'я, по батькові)

Перевірив Долголенко Олександр Миколайович
(прізвище, ім'я, по батькові)

Київ 2024

Лабораторна робота 3

Використання виконувачів із пакету *java.util.concurrent*

$$C = MC \cdot B - MM \cdot D;$$

$$MF = min(B + D) \cdot MC \cdot MZ + MM \cdot (MC + MM) \cdot a,$$

де *B*, *C*, *D* – вектори; *MC*, *MM*, *MF*, *MZ* – матриці; *a* – скаляр;

Спочатку наведемо всі виконувачі, які містяться у пакеті java.util.concurrent та підходять для цієї лабораторної роботи:

- **Executor** – базовий інтерфейс, який представляє об'єкт, що виконує надані йому задачі.
- **ExecutorService** – розширений інтерфейс, що надає методи для управління життєвим циклом, як завершення виконання та методи для відстеження статусу виконання задач.
- **ScheduledExecutorService** – варіант *ExecutorService*, який може виконувати задачі за розкладом або з певною періодичністю.
- **ThreadPoolExecutor** – одна з найбільш використовуваних реалізацій *ExecutorService*, що управляє пулом потоків для виконання задач.
- **ScheduledThreadPoolExecutor** – розширення *ThreadPoolExecutor* для підтримки запланованих і періодичних задач.

В нашій програмній реалізації немає випадків, коли доречно було б використовувати **Scheduled** методи, але їх реалізацію все одно продемонструємо.

Почнемо з використання **ExecutorService,** адже в ньому вже закладене використання звичайного **Executor**.

Використаємо ExecutorService для роботи з основними потоками. Їх 6, тому і початково визначимо їх як таку кількість:

```
private static final ExecutorService executor = Executors.newFixedThreadPool(6);
```

Для зручності подальшої роботи з виконувачами, перепишемо існуючі потоки як Runnable функції:

```
private static Runnable createSimpleMatrixTask(double[][] MC, double[][] MZ, double[]
B, double[] D, double[][] MM, double a, int amount) {
    return () -> {
        final CyclicBarrier barrier = new CyclicBarrier(3);
        double[][][] matrix = new double[2][][];
        double[][] MF;
        long startTime = System.currentTimeMillis();
        Thread tread1 = new Thread(() -> {
            try {
                matrix[0] = multiply(multiply(MC, MZ), min(add(B, D)));
                barrier.await();
            } catch (Exception e) {
                Thread.currentThread().interrupt();
                e.printStackTrace();
            }
        });// matrix[0] = multiply(multiply(MC, MZ), min(add(B, D))
        Thread tread2 = new Thread(() -> {
            try {
                matrix[1] = multiply(multiply(MM, add(MC, MM)), a);
                barrier.await();
            } catch (Exception e) {
                Thread.currentThread().interrupt();
                e.printStackTrace();
            }});// matrix[1] = multiply(multiply(MM, add(MC, MM)), a)
        tread1.start();
        tread2.start();
        try {
            barrier.await();
            MF = add(matrix[0], matrix[1]);
            timeMatrixSimple.set(System.currentTimeMillis() - startTime);
            synchronized (lock) {
                System.out.println("\nMatrix MF with modified threads:");
                output(MF, amount);
                writeToFile("MF_simple.txt", MF);
            }
        } catch (Exception e) {
            Thread.currentThread().interrupt();
            e.printStackTrace();}};}
```

```
private static Runnable createSimpleVectorTask(double[] B, double[][] MC, double[] D,
double[][] MM, int amount) {
    return () -> {
        long startTime = System.currentTimeMillis();
        double[] C = subtract(multiply(MC, B), multiply(MM, D));
        timeVectorSimple.set(System.currentTimeMillis() - startTime);

        synchronized (lock) {
            System.out.println("\n\nVector C with modified threads:");
            output(C, amount);
            writeToFile("C_simple.txt", C);
        }
    };
}

private static Runnable createKahanVectorTask(double[] B, double[][] MC, double[] D,
double[][] MM, int amount) {
    return () -> {
        long startTime = System.currentTimeMillis();
        double[] C_Kahan = subtract(multiplyKahan(MC, B), multiplyKahan(MM, D));
        timeVectorKahan.set(System.currentTimeMillis() - startTime);

        synchronized (lock) {
            System.out.println("\n\nVector C_Kahan with modified threads:");
            output(C_Kahan, amount);
            writeToFile("C_kahan.txt", C_Kahan);
        }
    };
}

private static Runnable createKahanMatrixTask(double[][] MC, double[][] MZ, double[]
B, double[] D, double[][] MM, double a, int amount) {
    return () -> {
        final CyclicBarrier barrier = new CyclicBarrier(3);
        double[][][] matrix = new double[2][][];
        double[][] MF_Kahan;
        long startTime = System.currentTimeMillis();
        Thread tread1 = new Thread(() -> {
            try {
                matrix[0] = multiply(multiplyKahan(MC, MZ), min(add(B, D)));
                barrier.await();
            } catch (Exception e) {
                Thread.currentThread().interrupt();
                e.printStackTrace();
            }
        });// matrix[0] = multiply(multiplyKahan(MC, MZ), min(addKahan(B, D))
```

```
        Thread tread2 = new Thread(() -> {
            try {
                matrix[1] = multiply(multiplyKahan(MM, add(MC, MM)), a);
                barrier.await();
            } catch (Exception e) {
                Thread.currentThread().interrupt();
                e.printStackTrace();
            }
        });// matrix[1] = multiply(multiplyKahan(MM, addKahan(MC, MM)), a)
        tread1.start();
        tread2.start();
        try {
            barrier.await();
            MF_Kahan = add(matrix[0], matrix[1]);
            timeMatrixKahan.set(System.currentTimeMillis() - startTime);
            synchronized (lock) {
                System.out.println("\nMatrix MF_Kahan with modified threads:");
                output(MF_Kahan, amount);
                writeToFile("MF_kahan.txt", MF_Kahan);
            }
        } catch (Exception e) {
            Thread.currentThread().interrupt();
            e.printStackTrace();
        }
    };
}

private static Runnable createKahanBabushkaVectorTask(double[] B, double[][] MC,
double[] D, double[][] MM, int amount) {
    return () -> {
        long startTime = System.currentTimeMillis();
        double[] C_KahanBabushka = subtract(multiplyKahanBabushka(MC, B),
multiplyKahanBabushka(MM, D));
        timeVectorKahanBabushka.set(System.currentTimeMillis() - startTime);

        synchronized (lock) {
            System.out.println("\n\nVector C_Kahan-Babushka with threads:");
            output(C_KahanBabushka, amount);
            writeToFile("C_kahan_babushka.txt", C_KahanBabushka);
        }
    };
}
```

```
private static Runnable createKahanBabushkaMatrixTask(double[][] MC, double[][] MZ,
double[] B, double[] D, double[][] MM, double a, int amount) {
    return () -> {
        final CyclicBarrier barrier = new CyclicBarrier(3);
        double[][][] matrix = new double[2][][];
        double[][] MF_KahanBabushka;
        long startTime = System.currentTimeMillis();
        Thread tread1 = new Thread(() -> {
            try {
                matrix[0] = multiply(multiplyKahanBabushka(MC, MZ), min(add(B, D)));
                barrier.await();
            } catch (Exception e) {
                Thread.currentThread().interrupt();
                e.printStackTrace();
            }
        });// matrix[0] = multiply(multiplyKahanBabushka(MC, MZ),
min(addKahanBabushka(B, D))
        Thread tread2 = new Thread(() -> {
            try {
                matrix[1] = multiply(multiplyKahanBabushka(MM, add(MC, MM)), a);
                barrier.await();
            } catch (Exception e) {
                Thread.currentThread().interrupt();
                e.printStackTrace();
            }
        });// matrix[1] = multiply(multiplyKahanBabushka(MM, add(MC, MM)), a)
        tread1.start();
        tread2.start();
        try {
            barrier.await();
            MF_KahanBabushka = add(matrix[0], matrix[1]);
            timeMatrixKahanBabushka.set(System.currentTimeMillis() - startTime);
            synchronized (lock) {
                System.out.println("\nMatrix MF_Kahan-Babushka with threads:");
                output(MF_KahanBabushka, amount);
                writeToFile("MF_kahan_babushka.txt", MF_KahanBabushka);
            }
        } catch (Exception e) {
            Thread.currentThread().interrupt();
            e.printStackTrace();
        }
    };
}
```

І тепер наш мейн метод буде виглядати так:

```
Runnable taskVectorSimple = createSimpleVectorTask(B, MC, D, MM, amount);
Runnable taskMatrixSimple = createSimpleMatrixTask(MC, MZ, B, D, MM, a, amount);
Runnable taskVectorKahan = createKahanVectorTask(B, MC, D, MM, amount);
Runnable taskMatrixKahan = createKahanMatrixTask(MC, MZ, B, D, MM, a, amount);
Runnable taskVectorKahanBabushka = createKahanBabushkaVectorTask(B, MC, D, MM,
amount);
Runnable taskMatrixKahanBabushka = createKahanBabushkaMatrixTask(MC, MZ, B, D, MM, a,
amount);

executor.execute(taskVectorSimple);
executor.execute(taskMatrixSimple);
executor.execute(taskVectorKahan);
executor.execute(taskMatrixKahan);
executor.execute(taskVectorKahanBabushka);
executor.execute(taskMatrixKahanBabushka);

executor.shutdown();

try {
    if (!executor.awaitTermination(60, TimeUnit.SECONDS)) {
        executor.shutdownNow();
    }
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
}
```

Щоб перевірити правильність нашого коду, переглянемо результати виводу:

```
Vector C_Kahan with modified threa
1481.1163911298463
399.7082351278914
1046.3426490022357
...
2189.3697116229414
1051.4603938456094
682.7540859878209
```

```
Vector C with modified thread
1481.1163911298245
399.708235127895
1046.3426490023267
...
2189.369711622916
1051.460393845642
682.7540859877263
```

```
Vector C_Kahan-Babushka with threa
1481.1163911298463
399.7082351278914
1046.3426490022357
...
2189.3697116229414
1051.4603938456094
682.7540859878209
```

І також вивід результуючої матриці різними алгоритмами:

```
Matrix MF with modified threads:
2.9742729015171297E305  2.9970516016261885E305  3.0315775027806355E305          ...
3.009162425369029E305   3.055265984206946 4E305          ...            3.013803476355965 6E305
2.948447647370917E305           ...             3.116709456718744E305   3.095118774353298E305
         ...            3.021496412562133 7E305  3.043806738300828E305   3.003179248852987 7E305

Matrix MF_Kahan with modified threads:
2.9742729015171317E305  2.9970516016261905E305  3.0315775027806374E305          ...
3.0091624253690296E305  3.055265984206947E305           ...            3.013803476355964 4E305
2.948447647370918E305           ...             3.116709456718747E305   3.095118774353295 3E305
         ...            3.021496412562131 7E305  3.043806738300824 2E305  3.003179248852984 6E305

Matrix MF_Kahan-Babushka with threads:
2.9742729015171317E305  2.9970516016261905E305  3.0315775027806374E305          ...
3.0091624253690296E305  3.055265984206947E305           ...            3.013803476355964 4E305
2.948447647370918E305           ...             3.116709456718747E305   3.095118774353295 3E305
         ...            3.021496412562131 7E305  3.043806738300824 2E305  3.003179248852984 6E305
```

Проаналізуємо швидкість виконання, але спочатку нагадаємо попередні результати вимірів:

```
Time vector simple = 11
Time matrix simple = 8207
Time vector Kahan = 13
Time matrix Kahan = 10118
Time vector Kahan-Babushka = 24
Time matrix Kahan-Babushka = 12265
```

Де сумарне виконання було за 12273мс. Тепер вкажемо поточні результати:

```
All time 12866
Time vector simple = 11
Time matrix simple = 9677
Time vector Kahan = 12
Time matrix Kahan = 10806
Time vector Kahan-Babushka = 27
Time matrix Kahan-Babushka = 12862
```

Як можемо побачити, різниці у часі майже немає, адже це все в межах похибки.

Тепер продемонструємо приклад використання ScheduledExecutorService. Код майже не зміниться, лише в main.

```
private static final ScheduledExecutorService scheduler =
Executors.newScheduledThreadPool(6);
```

```
scheduler.schedule(taskVectorSimple, 0, TimeUnit.MILLISECONDS);
scheduler.schedule(taskMatrixSimple, 0, TimeUnit.MILLISECONDS);
scheduler.schedule(taskVectorKahan, 0, TimeUnit.MILLISECONDS);
scheduler.schedule(taskMatrixKahan, 0, TimeUnit.MILLISECONDS);
scheduler.schedule(taskVectorKahanBabushka, 0, TimeUnit.MILLISECONDS);
scheduler.schedule(taskMatrixKahanBabushka, 0, TimeUnit.MILLISECONDS);
scheduler.shutdown();
try {
    if (!scheduler.awaitTermination(60, TimeUnit.SECONDS)) {
        scheduler.shutdownNow();
    }
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
}
```

Тобто ми просто замінили executor на scheduler та затримку виставили на 0мс, щоб вони виконувались одночасно. Також продемонструємо результати виконання по часу, але різниця також має бути мінімальна, адже прискорень роботи в цих методах ми не застосовували.

```
All time 11932
Time vector simple = 11
Time matrix simple = 7988
Time vector Kahan = 14
Time matrix Kahan = 10022
Time vector Kahan-Babushka = 28
Time matrix Kahan-Babushka = 11926
```

Так само ми могли б продемонструвати роботу з використанням ThreadPoolExecutor, але для ініціалізації конструктора потрібно 5 змінних: int corePoolSize, int maximumPoolSize, long keepAliveTime, TimeUnit unit, *BlockingQueue<Runnable> workQueue*. Можемо побачити, що тут присутня також *BlockingQueue*, викориистання якої є завданням 6ї лабораторної, тому поки не будемо використовувати таку реалізацію.

Отже, при виконанні лабораторної ми трохи змінили логіку виконання потоків, використавши Executor, ExecutorService та ScheduledExecutorService. Проаналізувавши результати ми впевнились, що програма виконується правильно та коректно видає результат. Заміряли час для кожного рішення та виявилося, що всі зміни знаходяться в 0.5 секундах від замірів з попередньої лабораторної, але це можна вважати в межах похибки, адже між використанням ExecutorService та ScheduledExecutorService не мало б бути великої різниці.