Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського» Інститут прикладного системного аналізу
Кафедра системного проектування

ЗВІТ

про виконання лабораторної роботи №2 з дисципліни
«Паралельні обчислення»

Виконав:
Студент III курсу
Групи ДА-92
Шляхов Данило Сергійович
Варіант №23

Київ – 2022

1. **Мета роботи:** розробка і реалізація паралельного алгоритму для задач із паралелізмом даних.
2. **Склад робочого місця:**

   - Обладнання: IBM-сумісний персональний комп'ютер.
   - Програмне забезпечення: операційна система Windows, Java SDK версії 1.2.2 або вище.

3. **Завдання:** Розробити програму, яка за допомогою бібліотеки Atomic і методу compareAndSet виконує наступні операції для одновимірного массиву (для потоків використовувати ExecutorService або parallelStream).
   a. Можливе виконання будь-якою мовою програмування з використанням відповідних конструкцій неблокуючих алгоритмів.
   b. Обов'язковим є наявність програмної реалізації порівняння з послідовним рішенням та з паралельним рішенням, що використовує блокування.
   c. Розмір масиву брати за формулою 100000*N, де N – номер вашого варіанту.
4. **Завдання за варіантом 23:**
   Кількості елементів кратних 17

Хід роботи

Для виконання роботи була обрана мова програмування GoLang.
Паралельні алгоритми використовують 4 потоки.
Паралельні алгоритми ділять масив на сторінки і кожен потік підраховує кількість чисел кратних 17 на своїй сторінці.

Задаємо розмір масиву та ініціалізуємо глобальні змінні.

```
const (
    sizeMultiplier = 100000
    N              = 23
    multipleOf     = 17
    test           = 0
)

var (
```

```
    arraySize = sizeMultiplier * N
)
```

Заповнення масиву.

```
1. func FillArray(array []int64) {
2.     s1 := rand.NewSource(time.Now().UnixNano())
3.     r := rand.New(s1)
4.     for i := range array {
5.         array[i] = r.Int63()
6.     }
7. }
```

Підрахунок чисел кратних 17 в однопоточному алгоритмі:

```
func CountMultiples(multipleOf int64, array []int64) int64 {
    var sum int64 = 0
    for _, v := range array {
        if v%multipleOf == 0 {
            sum++
        }
    }
    return sum
}
```

Тестова функція для перевірки корректності роботи:

```
func TestCounter(t *testing.T) {
    array := []int64{17, 15, 34, 1003, 2, 129359, 800, 9282}
    var result int64 = 4
    var multipleOf int64 = 17

    var testResult int64 = CountMultiples(multipleOf, array)

    if testResult != result {
        t.Error(testResult, " != ", result)
    }
}
```

Підрахунок в блокуючому алгоритмі (використовується mutex):

```
func CountMultiples(multipleOf int64, array []int64) int64 {
    var sum int64 = 0
    var page int = len(array) / threadCount
    var wg sync.WaitGroup
    var mu sync.Mutex
    for i := 0; i < len(array); {
        var high int
        if i+page < len(array)-1 {
            high = i + page
        } else {
            high = len(array) - 1
        }
        wg.Add(1)
        go func(low, high int) {
            defer wg.Done()
            for k := low; k < high; k++ {
                if array[k]%multipleOf == 0 {
                    mu.Lock()
                    sum++
                    mu.Unlock()
                }
            }
        }(i, high)
        i += page
    }
    wg.Wait()
    return sum
}
```

Підрахунок в неблокуючому алгоритмі (використовується атомарна функція atomic.AddInt63())

```
func CountMultiples(multipleOf int64, array []int64) int64 {
    var sum int64 = 0
    var page int = len(array) / threadsCount
    var wg sync.WaitGroup
    for i := 0; i < len(array); {
        var high int
        if i+page < len(array)-1 {
            high = i + page
        } else {
            high = len(array) - 1
        }
        wg.Add(1)
        go func(low, high int) {
```

```
            defer wg.Done()
            for k := low; k < high; k++ {
                if array[k]%multipleOf == 0 {
                    atomic.AddInt64(&sum, 1)
                }
            }
        }(i, high)
        i += page
    }
    wg.Wait()
    return sum
}
```

Якщо результати алгоритмів не співпадають, вони повернуть в консоль помилку.

```
    if single.CountMultiples(multipleOf, array) !=
nonblocking.CountMultiples(multipleOf, array) {
        log.Fatalln("error when comparing (nonblocking): ",
single.CountMultiples(multipleOf, array), " != ",
nonblocking.CountMultiples(multipleOf, array))
    }
    if single.CountMultiples(multipleOf, array) !=
blocking.CountMultiples(multipleOf, array) {
        log.Fatalln("error when comparing (blocking): ",
single.CountMultiples(multipleOf, array), " != ",
blocking.CountMultiples(multipleOf, array))
    }
```

Порівняння швидкості виконання різних алгоритмів.

start

single: 135037

27.9947ms

blocking: 135037

17.9684ms

nonblocking: 135037

11.9986ms

Посилання на GitHub репозиторій: Лаба3

Висновок

Під час виконання роботи була розроблена програма для вирішення задачі послідовним, паралельним з блокуванням та паралельним неблокуючим алгоритмами. Для паралельного з блокуванням був використаний mutex, що блокував лічильник. Для паралельного без блокування була використана атомарна функція atomic AddInt63(). За результатами порівняння алгоритмів, очевидно, що неблокуючий алгоритм швидше виконується, тому завжди варто використовувати атомарні функції де це можливо.