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 №4 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 з використанням механізмів асинхронних обчислень.
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 Java класу CompletableFuture та відповідних методів асинхронних розрахунків (supplyAsync(), thenApplyAsync() та ін.) виконує завдання згідно варіанту.

   Можливе виконання мовою програмування за власним вибором з використанням відповідних конструкцій асинхронних алгоритмів.
4. **Завдання за варіантом 23:**
   Створити 3 масиви (або колекції) з випадковими числами. У першому масиві - елементи помножити на 2. У другому - залишити тільки парні. У третьому - залишити елементи в діапазоні від 0.3 до 0.6 максимального значення. Відсортувати масиви і злити в один масив елементи, поділивши всі значення на 2.

Хід роботи

Для виконання роботи була обрана мова програмування GoLang.

В мові GoLang для початку виконання асинхронних обчисленнь використовується конструкція **go function()**, де function() – це функція, котра буде виконуватися асинхронно.

Для того, щоб отримати результат обчислень в майбутньому, за аналогом класу CompletableFuture, використовується така конструкція:

```
future := func() chan type {
    f := make(chan type)
    go func() {
```

```
        //…do something
        f <- result
}()
    return f
}()
result <- future
```

future – канал з якого можна буде зчитати результат асинхронних обчислень. Цей канал повертається з лямбда функції. В самій функції цей канал створюється, запускається горутина та повертається канал. В горутині проводяться обчислення і результат записується в канал f. В останньому рядку результат обчислень зчитується в змінну result.

Розглянемо це на практиці. Нижче приведений код з функції main. В даному шматочку коду показаний виклик функції SortFuture яка відсортує ar1 асинхронно. Результат можна отримати з каналу f1.

```
f1 = SortFuture(ar1)
ar1 = <-f1
```

Нижче приведена реалізація SortFuture()

```
func SortFuture(array []int64) chan []int64 {
    future := make(chan []int64)
    go func() {
        sort.Slice(array, func(i, j int) bool {
            return array[i] < array[j]
        })
        future <- array
    }()
    return future
}
```

Робота програми:

Filled arr3:

[49 1 15 2 4]

Filled arr1:

[1 12 34 49 30]

Sorted arr3:

[1 2 4 15 49]

Filled arr2:

[30 26 17 2 1]

Sorted arr1:

[1 12 30 34 49]

Left range [14 29] arr3:

[15]

Divided by 2 arr3:

[7]

Sorted arr2:

[1 2 17 26 30]

Multiplied by 2 arr1:

[2 24 60 68 98]

Divided by 2 arr1:

[1 12 30 34 49]

Removed odd arr2:

[2 26 30]

Divided by 2 arr2:

[1 13 15]

Arr1 after operations:

[1 12 30 34 49]

Arr2 after operations:

[1 13 15]

Arr3 after operations:

[7]

Dumped into shared array

[1 12 30 34 49 1 13 15 7]

Перевіримо корректність роботи.

Згенеровані колекції:

1. *[1 12 34 49 30]*
2. *[30 26 17 2 1]*
3. *[49 1 15 2 4]*

Колекції корректно відсортовані:

1. *[1 12 30 34 49]*
2. *[1 2 17 26 30]*
3. *[1 2 4 15 49]*

Успішно проведені операції над масивами, а саме:

1. Множення на 2
   [2 24 60 68 98]
2. Прибирання непарних
   [2 26 30]
3. Залишення в діапазоні [0.3, 0.6] від максимуму ([14, 29] в даному випадку)
   [15]

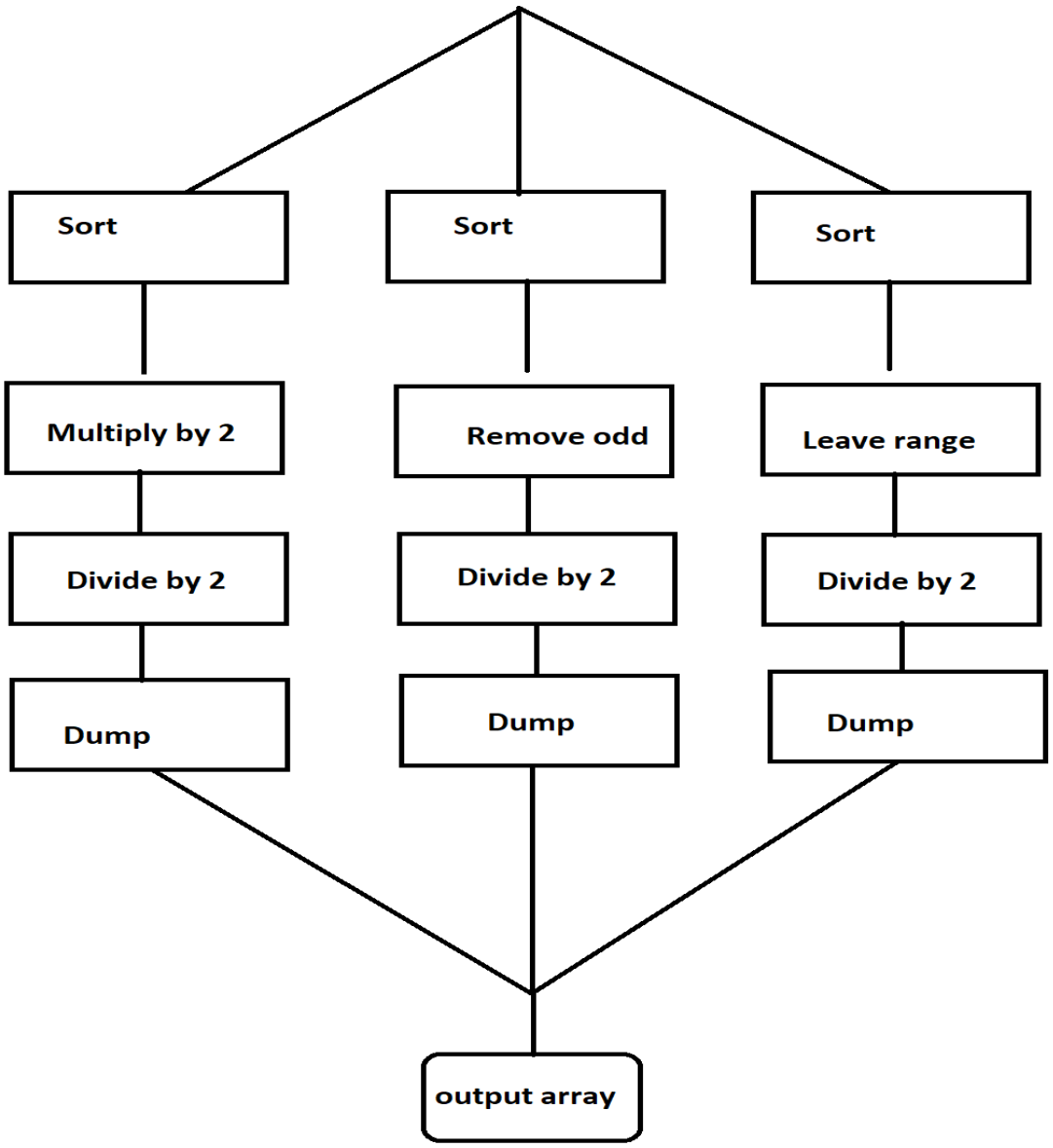
Всі елементи масивів були поділені на 2:

1. *[1 12 30 34 49]*
2. *[1 13 15]*
3. *[7]*

Після виконання всіх операцій елементи були злиті в один масив:

[1 12 30 34 49 1 13 15 7]

Діаграма роботи програми (намальовано в Paint):



Посилання на репозиторій: ЛАБА 4

Висновки:

В ході виконання лабораторної роботи було розроблено алгоритм дії та саму програму за варіантом за допомогою інструментів мови програмування golang. Було розглянуто можливості бібліотеки `CompletableFuture` для асинхронних операцій та застосовано на практиці аналоги наступних методів:

- *`supplyAsync()`*
- *`thenApplyAsync()`*
- *`thenCombine()`*

- *`get()`*