Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería de Sistema e Industrial
Computación Paralela y Distribuida
Ejercicio 1

Paralelismo basado en tareas (Fork-Join)

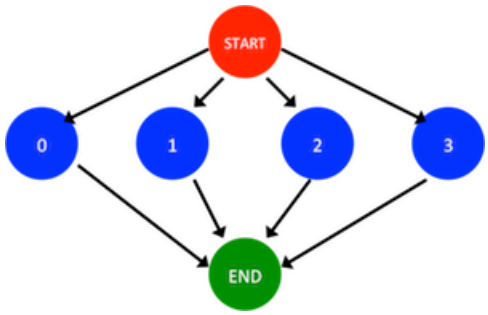
Para el ejercicio_1 se debe utilizar el framework Fork Join de Java para escribir una implementación paralela de la suma de los recíprocos de cada elemento de un arreglo.

El cálculo de la suma de los recíprocos de un arreglo implica sumar todos los elementos del arreglo. El recíproco de un valor v es 1/v.

El siguiente seudocódigo ilustra una manera secuencial para calcular la suma de los recíprocos de un arreglo A:

```
sum = 0
for v in A:
    sum = sum + (1 / v)
print ‘The reciprocal array sum of the input array is ‘ + sum
```

Es importante observar que el cálculo del recíproco en cada paso de la iteración es independiente del cálculo del recíproco de cualquier otra iteración, de tal forma que el problema puede paralelizarse. En particular el grafo computacional, para un arreglo con cuatro elementos puede ser visualizado como



Donde el círculo rojo indica el inicio del programa paralelo, el círculo verde la finalización y cada círculo azul representa una iteración del ciclo for.

La meta de este proyecto es utilizar el framework para fork-join de Java para paralelizar la implementación secuencial proporcionada.

I. Configuración del proyecto

Por favor, vea la documentación del ejercicio_0 para una descripción del proceso para construir y hacer las pruebas.

Descomprima el archivo ejercicio_1.zip anexo. Debe revisar el código fuente de

`Ejercicio_1/src/main/java/co/edu/unal/paralela/ReciprocalArraySum.java`

Y la prueba del proyecto

`Ejercicio_1/src/main/java/co/edu/unal/paralela/ReciprocalArraySumTest.java`

1. Instrucciones

Las modificaciones deben realizarse únicamente dentro del archivo `ReciprocalArraySum.java`. NO se deben cambiar las '*signatures*' de los métodos públicos (`public`) ni protegidos (`protected`) dentro de `ReciprocalArraySum.java`, pero se pueden editar el cuerpo de los métodosy adicional nuevos métodos si lo considera.

II. Actividades a realizar

1. Modificar el método `ReciprocalArraySum.parArraySum()` para implementar el cálculo de la suma de los recíprocos de un arreglo utilizando el framework Fork Join de Java mediante la partición del arreglo de entrada a la mitad y calculando la suma de los recíprocos del arreglo a partir de la suma de la primera parte y de la segunda parte en paralelo, antes de combinar los resultados. Hay varias actividades "Para hacer" (*TODOs*) en el código fuente como guía.

   La siguiente actividad es una generalizacoón de la primera, así que ya se debe tener confianza para particionar el trabajo entre múltiples tareas Fork-Join (no solo entre dos tareas).

   Observe que para realizar esta actividad y la siguiente se puede tener una ForkjoinPool dentro de parArraySum() y de parManyTaskArraySum() para ejecutar dentro las diferentes tareas. Por ejemplo, para crear una ForkJoinPool con 2 hilosse requiere el siguiente código:

   ```
   import java.util.concurrent.ForkJoinPool;

   ForkJoinPool pool = new ForkJoinPool(2);
   ```

2. Modificar el método `ReciprocalArraySum.parManyTaskArraySum` para implementar el cálculo de la suma de los recíprocos de un arreglo en

paralelo utilizando el framework Fork Join de Java nuevamente, pero esta vez se debe usar un número de tareas estipulado (no solo 2 tareas como en la actividad 1). Observe que los métodos `getChunkStartInclusive` y `getChunkEndExclusive` se presentan aquí para ayudar a calcular la región del arreglo de entrada que cierta tarea debe procesar.

III. Anexos

Ejemplo código de los videos del curso

```
public static double seqArraySum(double[] X) {
	long startTime = System.nanoTime();
	double sum = 0;
	for (int i=0; i < X.length; i++) {
		sum += 1/X[i];
	}
	long timeInNanos = System.nanoTime() - startTime;
	printResults("seqArraySum", timeInNanos, sum);
	// Tarea T0 espera por la tarea T1 (join)
	return sum;
}

/**
 * <p>parArraySum.</p>
 *
 * @param X una arreglo tipo double
 * @return sum of 1/X[i] for 0 <= i < X.length
*/
public static double parArraySum(double[] X) {
	long startTime = System.nanoTime();
	SumArray t = new SumArray(X, 0, X.length);
	ForkJoinPool.commonPool().invoke(t);
	double sum = t.ans;
	long timeInNanos = System.nanoTime() - startTime;
	printresults("parArraySum", timeInNanos, sum);
	return sum;
}

/**
 * <p>pmain</p>
 *
 * @param argv an array of double
 */
public static void main(final String[] argv) {
	// Initialization
	int n;
	if(argv.length !=0) {
		try {
			n = integer.parseInt(argv[0]);
			if(n <=0 ) {
```

```
                        // valor incorrecto de n
                        system.out.println(ERROR_MSG);
                        n = DEFAULT_N;
                }
        } catch (Throwable e) {
                System.out.println(ERROR_MSG);
                n = DEFAULT_N;
        }
    } else { // argv.length == 0
        n = DEFAULT_N;
    }
    double[] X = new double[n];

    for(int i=0; i <n; i++) {
        X[i] = (i + 1);
    }

    // establece el número de 'workers' utilizados por
ForKJoinPool.commonPool()
    System.setProperty("java.util.concurrent.ForkJoinPool.common.parallelism"
, "2");

    for (int numRun = 0; numRun < 5; numRun++) {
        System.out.printf("Run %d\n",numRun);
        seqArraySum(X);
        parArraySum(X);
    }
}

private static void printResults (String name, long timeInNanos, double sum) {
    system.out.printf(" %s completed in %8.3f milliseconds, with sum = %8.5f
\n", name, timeInNanos / 1e6, sum);
}

private static class SumArray extends recursiveAction {
    static int SEQUENTIAL_THRESHOLD = 5;
    int lo;
    int hi;
    double arr[];
    double ans = 0;

    SumArray(double[] a, int l, int h) {
        lo =l;
        hi = h;
        arr = a;
    }

    protected void compute() {
        if (hi - lo <= SEQUENTIAL_THRESHOLD) {
                for (int = lo; i < hi; ++i)
                        ans += 1 / arr[i];
        } else {
```

```
                    SumArray left = new SumArray(arr, lo, (hi + lo) /2);
                    SumArray right = new SumArray(arr, lo, (hi + lo) /2);
                    left.fork;
                    right.compute();
                    left.join();
                    ans = left.ans + right.ans;
                }
        }
}
```