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

## Ejercicio 4

**Sincronización de flujo de datos y Pipelining**

En este ejericio de deben aplicar phasers de java e hilos de java a un algoritmo de promedio iterativo uni-dimensional.

Descargar el archivo http://www.arcesio.net/paralela/ejercicio_4.zip

Las modificaciones solo se deben realizar en OneDimAveragingPhaser.java. No se pueden hacer cambios en las firmas (signatures) de los métodos públicos (public) ni en los protegidos (protected) dentro de OneDimAveragingPhaser, ni se pueden borrar. Se pueden adicionar métodos nuevos si se desea

OneDimAveragingPhaser.java contiene una actividad PARA HACER:

Basado en la version secuencial proporcionada en OneDimAveragingPhaser.runSequential y la version paralela en OneDimAveragingPhaser.runParallelBarrier, implementar una version paralela que utilice phasers para maximizar la superposición entre la terminación de la barrera y la realización de trabajo útil.

```
public static void seqMatrixMultiply(double[][] A, double B[][], double[][] C, int n) throws
SuspendableException {
      long startTime = System.nanoTime();
      forseq(0, n-1, 0, n-1, (i,j) ->
            {
                  C[i][j] = 0;
                  for (int k =0; k < n; k++) {
                        C[i][j] += A[i][k] * B[k][j];
                  }
            }
      );
      long timeInNanos = System.nanoTime() - startTime;
      printResults("seqMatrixMultiply", timeInNanos, C[n-1][n-1]);
}

public static void parMatrixMultiply(double[][] A, double B[][], double[][] C, int n) throws
SuspendableException {
      long startTime = System.nanoTime();
      forall(0, n-1, 0, n-1, (i,j) ->
            {
                  C[i][j] = 0;
                  for (int k =0; k < n; k++) {
                        C[i][j] += A[i][k] * B[k][j];
                  }
            }
      );
      long timeInNanos = System.nanoTime() - startTime;
      printResults("seqMatrixMultiply", timeInNanos, C[n-1][n-1]);
}
```

```
public static void parMatrixMultiplyChunked(double[][] A, double B[][], double[][] C, int n) throws
SuspendableException {
      long startTime = System.nanoTime();
      forallChunked(0, n-1, 0, n-1, (i,j) ->
            {
                  C[i][j] = 0;
                  for (int k =0; k < n; k++) {
                        C[i][j] += A[i][k] * B[k][j];
                  }
            }
      );
      long timeInNanos = System.nanoTime() - startTime;
      printResults("seqMatrixMultiply", timeInNanos, C[n-1][n-1]);
}

private static void printResults(string name, long timeInNanos, double sum) {
      System.out.printf(" %s completed in %8.3f milliseconds, with C[n-1][n-1] = %8.5f \n", name,
timeInNanos / 1e6, sum);
}
```