

APELLIDOS Y NOMBRES: Huamani Valentin Eladio Michael CÓDIGO: 20224025A

**Programación Paralela**

PRACTICA CALIFICADA Nº 1

20/09/2024

1. Efectuar un programa serial y paralelo (multi-threading para calcular la n-potencia de una matriz cuadrada ANxN. Determinar las salidas y tiempos correspondientes.

Archivo : PowerMatrix.java

public class PowerMatrix {

private Matrix A;

private final int N = 100;

private final int HILOS = 5;

private long[] times = **new** long[HILOS];

private long time ;

private long timeP ;

public static void main(String[] args) {

PowerMatrix pm = **new** PowerMatrix();

int n = 50;

Matrix C1 = pm.powerSerial(n);

Matrix C2 = pm.powerParallel(n);

System.out.println("Tiempo paralelo :" + pm.timeP);

for (int i = 0; i < pm.HILOS; i++) {

System.out.println("\tTiempo del hilo " + i + ": " + pm.times[i]);

}

System.out.println("Tiempo estatico :" + pm.time);

DataSet.WriteFile(C2, "resultadoParalelo.txt");

DataSet.WriteFile(C1, "resultado.txt");

}

public PowerMatrix(){

DataSet.CreateFile(N, N);

this.A = **new** Matrix(DataSet.ReadFile(N , N));

}

public Matrix powerSerial(int n){

time = System.currentTimeMillis();

if(n==0){

return **new** Matrix(**new** double[A.getRows()][A.getCols()]).toIdentity();

}

if(n==1){

return A;

}

Matrix C = **new** Matrix(A).toIdentity();

for (int i = 0; i < n ; i++) {

C = C.prod(A);

}

time = System.currentTimeMillis() - time;

return C;

}

public Matrix powerParallel(int n){

timeP = System.currentTimeMillis();

if(n == 0) {

return **new** Matrix(**new** double[A.getRows()][A.getCols()]).toIdentity();

}

if(n == 1) {

return A;

}

Matrix[] C = **new** Matrix[HILOS];

int[] pts = **new** int[HILOS];

int part = n / HILOS;

int rest = n % HILOS;

for (int i = 0; i < HILOS; i++) {

C[i] = **new** Matrix(A);

pts[i] = part;

if (i == HILOS - 1) {

pts[i] += rest;

}

}

Thread[] threads = **new** Thread[HILOS];

for (int t = 0; t < HILOS; t++) {

final int index = t;

threads[t] = **new** Thread(**new** Runnable() {

@Override

public void run() {

times[index] = System.currentTimeMillis();

Matrix temp = **new** Matrix(A).toIdentity();

for (int i = 0; i < pts[index]; i++) {

temp = temp.prod(A);

}

C[index] = temp;

times[index] = System.currentTimeMillis() - times[index];

}

});

threads[t].start();

}

try {

for (int t = 0; t < HILOS; t++) {

threads[t].join();

}

} catch (InterruptedException e) {

System.out.println(e.getMessage());

}

Matrix result = **new** Matrix(A).toIdentity();

for (Matrix matrix : C) {

result = result.prod(matrix);

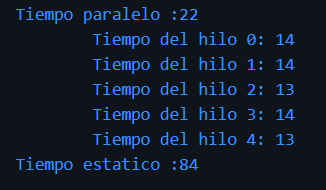
}

timeP = System.currentTimeMillis() - timeP;

return result;

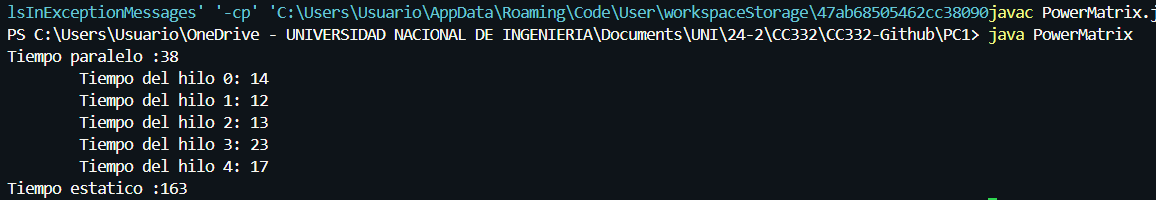
}

}



Salidas en los archivos result y resultParallel

1. Sea P el número de hilos. Dado el problema anterior, efectuar el procesamiento en cada k-ésimo hilo, 1≤k≤P. para cada hilo determinar el tiempo de ejecución TK y la salida correspondiente FK. Realizar las estadísticas correspondientes respecto al procesamiento total serial y paralelo del ejemplo anterior.



1. Efectuar un programa serial y paralelo (multi-threading ) para realizar la búsqueda secuencial de elementos (datos) almacenados en un archivo DATOS.TXT.

Archivo: Busqueda.java

public class Busqueda {

private Matrix A;

private final int N = 100;

private final int HILOS = 3;

private long[] times = **new** long[HILOS];

private long time;

private long timeP;

public static void main(String[] args) {

Busqueda bsq = **new** Busqueda();

double target = 4.97;

int c1 = bsq.search(target);

int c2 = bsq.searchParallel(target);

System.out.println("Tiempo paralelo: " + bsq.timeP);

for (int i = 0; i < bsq.HILOS; i++) {

System.out.println("\tTiempo del hilo " + i + ": " + bsq.times[i]);

}

System.out.println("\tEncontrado en la posición : " + c2 + "indice: ("+ c2/bsq.N+ ","+c2%bsq.N+")" );

System.out.println("Tiempo serial: " + bsq.time);

System.out.println("\tEncontrado en la posición : " + c1 + "indice: ("+ c1/bsq.N+ ","+c1%bsq.N+")" ) ;

}

public Busqueda() {

this.A = **new** Matrix(DataSet.ReadFile(N, N));

}

public int search(double target) {

time = System.currentTimeMillis();

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (A.GetCell(i, j) == target) {

time = System.currentTimeMillis() - time;

return i \* N + j;

}

}

}

time = System.currentTimeMillis() - time;

return -1;

}

public int searchParallel(double target) {

timeP = System.currentTimeMillis();

Thread[] threads = **new** Thread[HILOS];

int[] result = **new** int[HILOS];

int part = N / HILOS;

int rest = N % HILOS;

for (int i = 0; i < HILOS; i++) {

result[i] = -1;

}

for (int t = 0; t < HILOS; t++) {

final int hilo = t;

final int ini = t \* part;

final int fin = (t == HILOS - 1) ? N : ini + rest;

threads[t] = **new** Thread(**new** Runnable() {

@Override

public void run() {

times[hilo] = System.currentTimeMillis();

for (int i = ini; i < fin; i++) {

for (int j = 0; j < N; j++) {

if (A.GetCell(i, j) == target) {

result[hilo] = i \* N + j;

return;

}

}

}

times[hilo] = System.currentTimeMillis() - times[hilo];

}

});

threads[t].start();

}

try {

for (int t = 0; t < HILOS; t++) {

threads[t].join();

}

} catch (InterruptedException e) {

e.printStackTrace();

}

for (int t = 0; t < HILOS; t++) {

if (result[t] != -1) {

timeP = System.currentTimeMillis() - timeP;

return result[t];

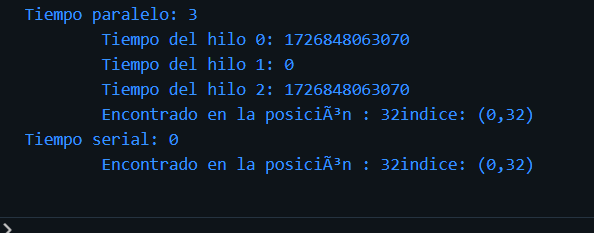
}

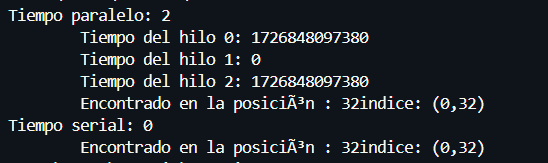
}

timeP = System.currentTimeMillis() - timeP;

return -1; }

}





1. Dado un problema computacional, efectuar el algoritmo, programa y grafo de dependencias asociado.