| EI1024/MT1024 “Programación Concurrente y Paralela” 2022–23  Nombre y apellidos (1)**Belen Ariño Bolinches**  Nombre y apellidos (2):Aleix Grau Bas  Tiempo empleado para tareas en casa en formato *h:mm* (obligatorio): . . . . . . . . . . . . . | Entregable  para  Laboratorio  la02 g |
| --- | --- |

Tema 02. Dise˜no de Algoritmos Paralelos

Tema 04. Conceptos Básicos de Concurrencia en Java

| 1 |
| --- |

Este ejercicio constituye una primera aproximación al ejercicio siguiente.

Se desea mostrar en pantalla todos los números comprendidos entre 0 y *n−*1 (incluyendo ambos extremos), donde *n* es un dato recibido en la línea de argumentos. El orden en el que se muestran los datos no es importante y se desea emplear varias hebras.

En la línea de comandos, el primer argumento será el número de hebras y el número *n* será el segundo. Si el número de argumentos de la línea de argumentos no es correcto, el programa debe avisar y terminar. Asimismo, si algún argumento no es correcto (por ejemplo, se esperaba un argumento numérico y no lo es), el programa también debe avisar y terminar.

Seguidamente se muestra la versión inicial de los códigos a implementar.

*// ============================================================================* c l a s s EjemploMuestraNumeros *{*

*// ============================================================================*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* public s t a t i c void main ( S t ri n g a r g s [ ] ) *{*

int n , numHebras ;

*// Comprobacion y e x t r a c c i o n de l o s argumen tos de en t r a d a .*

i f ( a r g s . l e n g t h != 2 ) *{*

System . e r r . p r i n t l n ( ”Uso : j a v a programa *<*numHebras*> <*n*>*” ) ;

System . e x i t ( −1 ) ;

*}*

try *{*

numHebras = I n t e g e r . p a r s e I n t ( a r g s [ 0 ] ) ;

n = I n t e g e r . p a r s e I n t ( a r g s [ 1 ] ) ;

*}* catch ( NumberFormatException ex ) *{*

numHebras = −1;

n = −1;

System . out . p r i n t l n ( ”ERROR: Argumentos nume ric o s i n c o r r e c t o s . ” ) ;

System . e x i t ( −1 ) ;

*}*

*//*

*// Implemen tac ion p a r a l e l a con d i s t r i b u c i o n c i c l i c a o por b l o q u e s . //*

*// Crea y arranca e l v e c t o r de h e b r a s .*

*// . . .*

*// Espera a que term inen l a s h e b r a s .*

*// . . .*

*}*

*}*

2

1.1) Implementa una versión paralela mediante el uso de hebras con una *distribución cíclica*. Realiza varias comprobaciones variando tanto el número de hebras como *n*, centrándote en los casos en los que *n* no es un múltiplo del número de hebras, por ejemplo *n* = 13 y *nH* = 4. Comprueba que aparecen todos los números deseados y que no hay repetidos. Escribe a continuaci´on la definici´on de la clase y el código a incluir en el programa principal.

// ============================================================================

class EjemploMuestraNumeros {

// ============================================================================

// --------------------------------------------------------------------------

class MiHebra extends Thread {

int numHebras;

int n;

int miId;

public MiHebra(int numHebras, int n,int miId){

this.numHebras = numHebras;

this.n = n;

this.miId = miId;

}

public void run ( ) {

for ( int i = miId ; i < n; i += numHebras) {

System.*out*.println(i) ;

}

}

}

public void main(String args[]) {

int n, numHebras;

// Comprobacion y extraccion de los argumentos de entrada.

if( args.length != 2 ) {

System.*err*.println( "Uso: java programa <numHebras> <n>" );

System.*exit*( -1 );

}

try {

numHebras = Integer.*parseInt*( args[ 0 ] );

n = Integer.*parseInt*( args[ 1 ] );

} catch( NumberFormatException ex ) {

numHebras = -1;

n = -1;

System.*out*.println( "ERROR: Argumentos numericos incorrectos." );

System.*exit*( -1 );

}

//

// Implementacion paralela con distribucion ciclica o por bloques.

MiHebra v[] = new MiHebra[numHebras];

for (int i = 0; i < numHebras; i++)

v[i] = new MiHebra(numHebras,n,i);

// Crea y arranca el vector de hebras.

for (int i = 0; i < v.length; i++)

v[i].start();

// Espera a que terminen las hebras.

for (int i = 0; i < v.length; i++){

try {

v[i].join();

}catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

}

}

3

1.2) Implementa una versi´on paralela mediante el uso de hebras con una *distribuci´on por bloques*. Realiza varias comprobaciones variando tanto el n´umero de hebras como *n*, centr´andote en los casos en los que *n* no es un m´ultiplo del n´umero de hebras, por ejemplo *n* = 13 y *nH* = 4. Comprueba que aparecen todos los n´umeros deseados y que no hay repetidos.

Escribe a continuaci´on la definici´on de la clase y el código a incluir en el programa principal.

class MiHebra extends Thread {

int numHebras;

int n;

int miId;

public MiHebra(int numHebras, int n,int miId){

this.numHebras = numHebras;

this.n = n;

this.miId = miId;

}

public void run ( ) {

for ( int i = miId \* (n/numHebras) ; i < (miId + 1) \* (n/numHebras) ; i++) {

System.*out*.println(i) ;

}

}

}

public void main(String args[]) {

int n, numHebras;

// Comprobacion y extraccion de los argumentos de entrada.

if( args.length != 2 ) {

System.*err*.println( "Uso: java programa <numHebras> <n>" );

System.*exit*( -1 );

}

try {

numHebras = Integer.*parseInt*( args[ 0 ] );

n = Integer.*parseInt*( args[ 1 ] );

} catch( NumberFormatException ex ) {

numHebras = -1;

n = -1;

System.*out*.println( "ERROR: Argumentos numericos incorrectos." );

System.*exit*( -1 );

}

//

// Implementacion paralela con distribucion ciclica o por bloques.

MiHebra v[] = new MiHebra[numHebras];

for (int i = 0; i < numHebras; i++)

v[i] = new MiHebra(numHebras,n,i);

// Crea y arranca el vector de hebras.

for (int i = 0; i < v.length; i++)

v[i].start();

// Espera a que terminen las hebras.

for (int i = 0; i < v.length; i++){

try {

v[i].join();

}catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

}

}

4

| 2 |
| --- |

Evaluaci´on de una funci´on en m´ultiples puntos.

Se dispone de un programa secuencial que eval´ua una funci´on en varios puntos. Para almacenar datos y resultados, el c´odigo emplea dos vectores de la misma dimensi´on: vectorX y vectorY. El c´odigo eval´ua la funci´on para cada elemento de vectorX dejando el resultado calculado en el correspondiente elemento de vectorY. Este tipo de c´alculo es muy habitual y se puede realizar por ejemplo para visualizar una funci´on en pantalla.

Tras realizar los c´alculos, en lugar de visualizar los resultados en pantalla, el programa realiza y muestra en pantalla la suma de los elementos de ambos vectores para que despu´es se puedan comprobar f´acilmente los resultados con las versiones paralelas.

El programa secuencial declara e inicializa una variable con el n´umero de hebras, pero no la emplea para nada. Dicho valor deber´a ser aprovechado en las implementaciones paralelas.

El c´odigo es el siguiente:

*// ============================================================================* c l a s s E jemploFuncionCostosa *{*

*// ============================================================================*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* public s t a t i c void main ( S t ri n g a r g s [ ] ) *{*

int n , numHebras ;

long t1 , t 2 ;

double sumaX , sumaY , t s , tc , tb ;

*// Comprobacion y e x t r a c c i o n de l o s argumen tos de en t r a d a .*

i f ( a r g s . l e n g t h != 2 ) *{*

System . e r r . p r i n t l n ( ”Uso : j a v a programa *<*numHebras*> <*tamanyo*>*” ) ;

System . e x i t ( −1 ) ;

*}*

try *{*

numHebras = I n t e g e r . p a r s e I n t ( a r g s [ 0 ] ) ;

n = I n t e g e r . p a r s e I n t ( a r g s [ 1 ] ) ;

*}* catch ( NumberFormatException ex ) *{*

numHebras = −1;

n = −1;

System . out . p r i n t l n ( ”ERROR: Argumentos nume ric o s i n c o r r e c t o s . ” ) ;

System . e x i t ( −1 ) ;

*}*

*// Crea l o s v e c t o r e s .*

double vectorX [ ] = new double [ n ] ;

double vectorY [ ] = new double [ n ] ;

*//*

*// Implemen tac ion s e c u e n c i a l ( s i n t em p o r i z a r ) .*

*//*

i n i c i a l i z a V e c t o r X ( vectorX ) ;

i n i c i a l i z a V e c t o r Y ( vectorY ) ;

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

vectorY [ i ] = ev alu aFunci on ( vectorX [ i ] ) ;

*}*

*//*

*// Implemen tac ion s e c u e n c i a l .*

*//*

i n i c i a l i z a V e c t o r X ( vectorX ) ;

i n i c i a l i z a V e c t o r Y ( vectorY ) ;

t 1 = System . nanoTime ( ) ;

5

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

vectorY [ i ] = ev alu aFunci on ( vectorX [ i ] ) ;

*}*

t 2 = System . nanoTime ( ) ;

t s = ( ( double ) ( t 2 − t 1 ) ) / 1. 0 e9 ;

System . out . p r i n t l n ( ”Tiempo s e c u e n c i a l ( s e g . ) : ” + t s ) ; *// // impr imeResul tado ( vec torX , vec t orY ) ;*

*// Comprueba e l r e s u l t a d o .*

sumaX = sumaVector ( vectorX ) ;

sumaY = sumaVector ( vectorY ) ;

System . out . p r i n t l n ( ”Suma d el v e c t o r X: ” + sumaX ) ;

System . out . p r i n t l n ( ”Suma d el v e c t o r Y: ” + sumaY ) ;

*/∗*

*//*

*// Implemen tac ion p a r a l e l a c i c l i c a .*

*//*

*i n i c i a l i z a V e c t o r X ( vec t orX ) ;*

*i n i c i a l i z a V e c t o r Y ( vec t orY ) ;*

*t 1 = System . nanoTime ( ) ;*

*// Ge s t i on de h e b r a s para l a im plemen t ac i on p a r a l e l a c i c l i c a*

*// . . . .*

*t 2 = System . nanoTime ( ) ;*

*t c = ( ( d o u bl e ) ( t 2 − t 1 ) ) / 1 . 0 e9 ;*

*System . ou t . p r i n t l n ( ”Tiempo p a r a l e l a c i c l i c a ( s e g . ) : ” + t c ) ; System . ou t . p r i n t l n ( ” Incremen to p a r a l e l a c i c l i c a : ” + . . . ) ; //// impr imeResul tado ( vec torX , vec t orY ) ;*

*// Comprueba e l r e s u l t a d o .*

*sumaX = sumaVector ( vec t orX ) ;*

*sumaY = sumaVector ( vec t orY ) ;*

*System . ou t . p r i n t l n ( ”Suma d e l v e c t o r X: ” + sumaX ) ;*

*System . ou t . p r i n t l n ( ”Suma d e l v e c t o r Y: ” + sumaY ) ;*

*//*

*// Implemen tac ion p a r a l e l a por b l o q u e s .*

*//*

*// . . . .*

*//*

*∗/*

System . out . p r i n t l n ( ”Fin d el programa . ” ) ;

*}*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* s t a t i c void i n i c i a l i z a V e c t o r X ( double vectorX [ ] ) *{*

i f ( vectorX . l e n g t h == 1 ) *{*

vectorX [ 0 ] = 0 . 0 ;

*}* e l s e *{*

for ( int i = 0 ; i *<* vectorX . l e n g t h ; i++ ) *{*

vectorX [ i ] = 1 0. 0 ∗ ( double ) i / ( ( double ) vectorX . l e n g t h − 1 ) ; *}*

*}*

*}*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* s t a t i c void i n i c i a l i z a V e c t o r Y ( double vectorY [ ] ) *{*

for ( int i = 0 ; i *<* vectorY . l e n g t h ; i++ ) *{*

vectorY [ i ] = 0 . 0 ;

*}*

*}*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* s t a t i c double sumaVector ( double v e c t o r [ ] ) *{*

double suma = 0 . 0 ;

for ( int i = 0 ; i *<* v e c t o r . l e n g t h ; i++ ) *{*

suma += v e c t o r [ i ] ;

*}*

return suma ;

*}*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* s t a t i c double ev alu aFunci on ( double x ) *{*

return Math . s i n ( Math . exp ( −x ) + Math . l o g 1 p ( x ) ) ;

*}*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* s t a t i c void imprimeVector ( double v e c t o r [ ] ) *{*

for ( int i = 0 ; i *<* v e c t o r . l e n g t h ; i++ ) *{*

System . out . p r i n t l n ( ” v e c t o r [ ” + i + ” ] = ” + v e c t o r [ i ] ) ;

*}*

*}*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* s t a t i c void imprimeResultado ( double vectorX [ ] , double vectorY [ ] ) *{* for ( int i = 0 ; i *<* Math . min ( vectorX . len g th , vectorY . l e n g t h ) ; i++ ) *{* System . out . p r i n t l n ( ” i : ” + i +

” x : ” + vectorX [ i ] +

” y : ” + vectorY [ i ] ) ;

*}*

*}*

*}*

7

2.1) Paraleliza el c´odigo anterior mediante el uso de hebras con una *distribuci´on c´ıclica*. Descomenta el c´odigo situado debajo de“Implementacion secuencial”. Incluye la gesti´on de hebras que paraleliza el bucle comprendido entre la lectura de t1 y t2 en la versi´on secuencial, y la expresi´on que permite calcular el incremento de velocidad.

Verifica en varios casos que el nuevo c´odigo devuelve el mismo resultado que el original. Escribe a continuaci´on la parte de tu c´odigo que realiza tal tarea: la definici´on de la clase hebra y el c´odigo a incluir en el programa principal que permite gestionar los objetos de esta clase, as´ı como la visualizaci´on de los resultados y prestaciones.

// ============================================================================

class EjemploFuncionCostosa {

// ============================================================================

static class MiHebra1 extends Thread{

int numHebras;

int n;

int miId;

double vectorX[];

double vectorY[];

private MiHebra1(int numHebras,int n, int miId, double vectorX[], double vectorY[]){

this.numHebras = numHebras;

this.n = n;

this.miId = miId;

this.vectorX = vectorX;

this.vectorY = vectorY;

}

public void run(){

for( int i = miId; i < n; i += numHebras ) {

vectorY[ i ] = *evaluaFuncion*( vectorX[ i ] );

}

}

}

// --------------------------------------------------------------------------

public static void main(String args[]) {

int n = 16;

int numHebras = 4;

long t1, t2;

double sumaX, sumaY, ts, tc, tb;

// Comprobacion y extraccion de los argumentos de entrada.

// Crea los vectores.

double vectorX[] = new double[ n ];

double vectorY[] = new double[ n ];

//

// Implementacion secuencial (sin temporizar).

//

*inicializaVectorX*( vectorX );

*inicializaVectorY*( vectorY );

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

vectorY[ i ] = *evaluaFuncion*( vectorX[ i ] );

}

// //

// // Implementacion secuencial.

// //

// inicializaVectorX( vectorX );

// inicializaVectorY( vectorY );

// t1 = System.nanoTime();

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

// vectorY[ i ] = evaluaFuncion( vectorX[ i ] );

// }

// t2 = System.nanoTime();

// ts = ( ( double ) ( t2 - t1 ) ) / 1.0e9;

// System.out.println( "Tiempo secuencial (seg.): " + ts );

// //// imprimeResultado( vectorX, vectorY );

// // Comprueba el resultado.

// sumaX = sumaVector( vectorX );

// sumaY = sumaVector( vectorY );

// System.out.println( "Suma del vector X: " + sumaX );

// System.out.println( "Suma del vector Y: " + sumaY );

//

// Implementacion paralela ciclica.

//

*inicializaVectorX*( vectorX );

*inicializaVectorY*( vectorY );

t1 = System.*nanoTime*();

// Gestion de hebras para la implementacion paralela ciclica

// ....

MiHebra1 v[] = new MiHebra1[numHebras];

for (int i = 0; i < numHebras; i++)

v[i] = new MiHebra1(numHebras,n,i,vectorX,vectorY);

for (int i = 0; i < numHebras; i++)

v[i].start();

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

try{

v[i].join();

} catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

t2 = System.*nanoTime*();

tc = ( ( double ) ( t2 - t1 ) ) / 1.0e9;

System.*out*.println( "Tiempo paralela ciclica (seg.): " + tc );

System.*out*.println( "Incremento paralela ciclica: " );

//// imprimeResultado( vectorX, vectorY );

// Comprueba el resultado.

sumaX = *sumaVector*( vectorX );

sumaY = *sumaVector*( vectorY );

System.*out*.println( "Suma del vector X: " + sumaX );

System.*out*.println( "Suma del vector Y: " + sumaY );

//

// Implementacion paralela por bloques.

//

// ....

//

System.*out*.println( "Fin del programa." );

}

// --------------------------------------------------------------------------

static void inicializaVectorX( double vectorX[] ) {

if( vectorX.length == 1 ) {

vectorX[ 0 ] = 0.0;

} else {

for( int i = 0; i < vectorX.length; i++ ) {

vectorX[ i ] = 10.0 \* ( double ) i / ( ( double ) vectorX.length - 1 );

}

}

}

// --------------------------------------------------------------------------

static void inicializaVectorY( double vectorY[] ) {

for( int i = 0; i < vectorY.length; i++ ) {

vectorY[ i ] = 0.0;

}

}

// --------------------------------------------------------------------------

static double sumaVector( double vector[] ) {

double suma = 0.0;

for( int i = 0; i < vector.length; i++ ) {

suma += vector[ i ];

}

return suma;

}

// --------------------------------------------------------------------------

static double evaluaFuncion( double x ) {

return Math.*sin*( Math.*exp*( -x ) + Math.*log1p*( x ) );

}

// --------------------------------------------------------------------------

static void imprimeVector( double vector[] ) {

for( int i = 0; i < vector.length; i++ ) {

System.*out*.println( " vector[ " + i + " ] = " + vector[ i ] );

}

}

// --------------------------------------------------------------------------

static void imprimeResultado( double vectorX[], double vectorY[] ) {

for( int i = 0; i < Math.*min*( vectorX.length, vectorY.length ); i++ ) {

System.*out*.println( " i: " + i +

" x: " + vectorX[ i ] +

" y: " + vectorY[ i ] );

}

}

}

. . . . . . . . . . . . . . . .

| ATENCION: Los ejercicios anteriores deben realizarse en casa. Los siguientes, en el aula. ´ |
| --- |

2.2) Paraleliza el anterior código mediante el uso de **hebras** con una ***distribución por bloques***. Replica el código de la “Implementación cíclica” y adapta la gestión de hebras. La nueva versión paralela debe ejecutarse tras las anteriores versiones.

Verifica en varios casos que el nuevo código devuelve el mismo resultado que el original. Escribe a contin uaci ́on la parte de tu código que realiza tal tarea: la definición de la clase hebra y el código a incluir en el programa principal que permite gestionar los objetos de esta clase, así como la visualización de los resultados y prestaciones.

static class MiHebra1 extends Thread{

int numHebras;

int n;

int miId;

double vectorX[];

double vectorY[];

private MiHebra1(int numHebras,int n, int miId, double vectorX[], double vectorY[]){

this.numHebras = numHebras;

this.n = n;

this.miId = miId;

this.vectorX = vectorX;

this.vectorY = vectorY;

}

public void run(){

for( int i = miId; i < n; i += numHebras ) {

vectorY[ i ] = *evaluaFuncion*( vectorX[ i ] );

}

}

}

// Implementacion paralela por bloques.

*inicializaVectorX*( vectorX2 );

*inicializaVectorY*( vectorY2 );

t1 = System.*nanoTime*();

// Gestion de hebras para la implementacion paralela ciclica

// ....

MiHebra2 v1[] = new MiHebra2[numHebras];

for (int i = 0; i < numHebras; i++)

v1[i] = new MiHebra2(numHebras,n,i,vectorX2,vectorY2);

for (int i = 0; i < numHebras; i++)

v1[i].start();

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

try{

v1[i].join();

} catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

t2 = System.*nanoTime*();

tc = ( ( double ) ( t2 - t1 ) ) / 1.0e9;

System.*out*.println( "Tiempo paralela por bloques (seg.): " + tc );

System.*out*.println( "Incremento paralela por bloques: " + (ts - tc));

//// imprimeResultado( vectorX, vectorY );

// Comprueba el resultado.

sumaX2 = *sumaVector*( vectorX2 );

sumaY2 = *sumaVector*( vectorY2 );

System.*out*.println( "Suma del vector X: " + sumaX2 );

System.*out*.println( "Suma del vector Y: " + sumaY2);

9

2.3) Evalúa los códigos en el ordenador del laboratorio.

Copia los resultados en las siguientes tablas redonde´andolos con dos decimales. Ten en cuenta que, para poder calcular los incrementos de velocidad correctamente, se deben realizar las dos versiones paralelas (c´ıclica y por bloques) en una misma ejecuci´on, comparando sus prestaciones con las de la versi´on secuencial.

Incrementos de velocidad para *n* = 1 000 000

| N´umero de hebras | 1 | 2 | 4 |
| --- | --- | --- | --- |
| Distribución c´ıclica | 0.87 | 1.58 | 1.94 |
| Distribución por bloques | 0.97 | 1.49 | 2.03 |

Incrementos de velocidad para *n* = 10 000 000

| N´umero de hebras | 1 | 2 | 4 |
| --- | --- | --- | --- |
| Distribuci´on c´ıclica | 1.00 | 1.90 | 3.69 |
| Distribuci´on por bloques | 0.99 | 1.89 | 3.66 |

Examina con detalle y justifica los resultados.

Vemos como con mas de una hebra encontramos un incremento de velocidad, mientras que en los casos en los que tenemos una sola hebra perdemos velocidad y los programas paralelos son mas lentos

2.4) Cuando la funci ́on e**valuaFuncion** del apartado anterior se aplica a un **vector de n´umeros**, ¿el código resultante está limitado por la CPU, por la memoria central o por la E/S?

static double evaluaFuncion( double x ) {

return Math.*sin*( Math.*exp*( -x ) + Math.*log1p*( x ) );

}

Esta función estará limitada por la CPU

2.5) Si se utilizase la funci´on **evaluaFuncion** que se muestra y se aplicase a un vector de n´umeros, entonces ¿el c´odigo estar´ıa limitado por la CPU, por la memoria central o por la E/S?

s t a t i c double ev aluaFunci on ( double x ) *{*

return 3. 5 ∗ x ;

*}*

En este caso la función estaría limitada por memoria central

2.6) Haz una copia del programa que acabas de completar y dale el nombre de **EjemploFuncionSencilla.java.** Modif´ıcalo para que emplee la nueva funci´on evaluaFunci´on m´as sencilla. A continuación, calcula con dos decimales los incrementos de velocidad correspondientes para completar las siguientes tablas.

Incrementos de velocidad para *n* = 1 000 000

| Número de hebras | 1 | 2 | 4 |
| --- | --- | --- | --- |
| Distribución cíclica | 0.55 | 0.37 | 0.13 |
| Distribución por bloques | 0.39 | 0.31 | 0.17 |

Incrementos de velocidad para *n* = 10 000 000

| Número de hebras | 1 | 2 | 4 |
| --- | --- | --- | --- |
| Distribución cíclica | 0.97 | 0.61 | 0.33 |
| Distribución por bloques | 0.97 | 0.74 | 0.70 |

Examina con detalle y justifica los resultados.

Observamos que no aumenta la velocidad del programa, esto es debido a que la simplicidad de la nueva funcion hace que el crear nuevas instancias de hebras y datos ralentice el programa

2.7) ¿Los dos códigos (EjemploFuncionCostosa y EjemploFuncionSencilla) obtienen incrementos aceptables? Comenta los resultados.

EjemploFuncionCostosa: El método evaluaFuncion estará limitado por la CPU, por lo tanto al ejecutarlo con la distribución cíclica y con bloques está limitación se reducirá y optimizaremos la ejecución del código consiguiendo incrementos aceptables.

EjemploFuncionSencilla: No obtiene incrementos aceptables ya que a mayor es el problema observamos que el incremento empieza a ser negativo, es decir, va más despacio

2.8) Evalúa los códigos en patan.

Calcula con dos decimales los incrementos de velocidad necesarios para completar las si guientes tablas.

Recuerda que, para poder calcular los incrementos de velocidad correctamente, se deben ejecutar las dos versiones paralelas (c´ıclica y por bloques) en una misma ejecuci´on, compa rando sus prestaciones con las de la versi´on secuencial.

Examina con detalle y justifica los resultados.

21761.patan.act.uji.es

Resultados para evaluaFuncionCostosa

Incrementos de velocidad para *n* = 100 000 000

| N´umero de hebras | 1 | 4 | 8 | 16 |
| --- | --- | --- | --- | --- |
| Distribuci´on c´ıclica |  |  |  |  |
| Distribuci´on por bloques |  |  |  |  |

Resultados para evaluaFuncionSencilla

Incrementos de velocidad para *n* = 100 000 000

| N´umero de hebras | 1 | 4 | 8 | 16 |
| --- | --- | --- | --- | --- |
| Distribuci´on c´ıclica |  |  |  |  |
| Distribuci´on por bloques |  |  |  |  |

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

| Nota: Esta entrega forma parte de la evaluaci´on de la asignatura. Debe ser guardado por el estudiantado junto con el resto de entregas en una carpeta. El profesorado podr´a pedir al estudiantado que le entregue dicha carpeta en cualquier momento. |
| --- |