s

| EI1024/MT1024 “Programaci´on Concurrente y Paralela” 2022–23  Nombre y apellidos (1): Belén Ariño Bolinches  Nombre y apellidos (2): Aleix Grau Bas  Tiempo empleado para tareas en casa en formato *h:mm* (obligatorio): . . . . . . . . . . . . . | Entregable  para  Laboratorio  la03 g |
| --- | --- |

Tema 05. El Problema de la Visibilidad en Java

Tema 06. El Problema de la Atomicidad en Java

| 1 |
| --- |

Estudia el siguiente código y responde a las siguientes preguntas.

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

c l a s s Cuenta Incrementos *{*

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

int numIncrementos = 0 ;

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*

void incrementaNumIncrementos ( ) *{*

numIncrementos++;

*}*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*

int dameNumIncrementos ( ) *{*

return ( numIncrementos ) ;

*}*

*}*

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

c l a s s MiHebra extends Thread *{*

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

int t ope ;

Cuenta Incrementos c ;

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* public MiHebra ( int tope , Cuenta Incrementos c ) *{*

th is . t ope = t ope ;

th is . c = c ;

*}*

*// −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−* public void run ( ) *{*

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

c . incrementaNumIncrementos ( ) ;

*}*

*}*

*}*

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

c l a s s E jemploCuenta Incrementos *{*

*// ============================================================================ // −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*

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

long t1 , t 2 ;

double t t ;

int numHebras , t ope ;

*// 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*> <*tope*>*” ) ;

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 ] ) ;

t ope = 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;

t ope = −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 ) ;

*}*

System . out . p r i n t l n ( ”numHebras : ” + numHebras ) ;

System . out . p r i n t l n ( ” t ope : ” + t ope ) ;

System . out . p r i n t l n ( ”Creando y a r r anc and o ” + numHebras + ” heb r a s . ” ) ; t 1 = System . nanoTime ( ) ;

MiHebra v [ ] = new MiHebra [ numHebras ] ;

Cuenta Incrementos c = new Cuenta Incrementos ( ) ;

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

v [ i ] = new MiHebra ( tope , c ) ;

v [ i ] . s t a r t ( ) ;

*}*

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

try *{*

v [ i ] . j o i n ( ) ;

*}* catch ( I n t e r r u p t e dE x c e p ti o n ex ) *{*

ex . p ri n t S t a c kT r a c e ( ) ;

*}*

*}*

t 2 = System . nanoTime ( ) ;

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

System . out . p r i n t l n ( ” T o t al de i n c r em e n t o s : ” + c . dameNumIncrementos ( ) ) ; System . out . p r i n t l n ( ”Tiempo t r a n s c u r r i d o en s e g s . : ” + t t ) ;

*}*

*}*

1.1) ¿Qué realiza el código? ¿Qué debería mostrar en pantalla si se ejecutase con los parámetros hebras 4 y tope 1 000 000?

El código crea tantas hebras como le indiquemos a las que les pasamos un entero (número de hebras) y un objeto de la misma clase (CuentaIncrementos).

Las almacena en un vector v y despúes las arranca (Crea y arranca de hebra en hebra)

Cada hebra se incrementa hasta el valor pasado como tope (incrementamos la variable c mediante **c.incrementaNumIncrementos()**

Por último muestra por pantalla el total de incrementos realizados y el tiempo en el que se ha realizado

1.2) Compila y ejecuta el código con dichos valores en tu ordenador local. ¿Que muestra realmente en pantalla si se ejecuta con los parámetros hebras 4 y tope 1 000 000?

numHebras: 4

tope: 1000000

Creando y arrancando 4 hebras

Total incrementos: 2779717

Tiempo transcurrido en segundos: 0.02313

3

1.3) ¿Es un código *thread-safe*? Justifica tu respuesta.

No lo es, el código tiene problemas de atomicidad, ya que la clase CuentaIncrementos es accedida sin ningún tipo de sincronización, lo que puede generar condiciones de carrera

1.4) Crea una copia del código original e inserta en la copia el modificador **volatile** en la variable numIncrementos de la clase CuentaIncrementos.

A continuación, compila y prueba el nuevo código. ¿Resuelve el problema el modificador volatile? ¿Por qué?

No soluciona el código, ya que volatile almacena la variable local en memoria central, pero no solucionará el problema de atomicidad, ya que la variable puede ser leída mientras otra hebra la está modificando.

1.5) ¿Se podría resolver con el modificador synchronized?

Para ello, crea una copia del código original y aplica el modificador synchronized sobre cada una de las rutinas de la clase CuentaIncrementos. Después compila y prueba el código, antes de contestar a la pregunta anterior. Escribe a continuación los cambios realizados en la clase CuentaIncrementos.

Al poner el modificador synchronized en los métodos de la clase CuentaIncrementos, mientras un hilo esté ejecutando uno de ellos no podrá acceder otro. Esto soluciona el problema de atomicidad pero en consecuencia el tiempo que tarda en ejecutarse aumenta mucho. 0.1355569s ( 6 veces más que el sin synchronized)

1.6) ¿Se puede arreglar empleando clases y operadores atómicos?

Para ello, crea otra copia del código original, ELIMINA COMPLETAMENTE la clase CuentaIncrementos y utiliza en su lugar una clase atómica y sus métodos. Después compila y prueba el código, antes de contestar la pregunta.

Escribe a continuación los cambios realizados en el código.

Creo una copia del código, en el cuál creamos una nueva variable AtomicInteger y la inicializamos a 0

class MiHebra extends Thread {

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

int tope;

AtomicInteger c;

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

public MiHebra( int tope, AtomicInteger c ) {

this.tope = tope;

this.c = c;

}

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

public void run() {

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

c.incrementAndGet();

}

}

}

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

class EjemploCuentaIncrementos {

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

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

public static void main( String args[] ) {

long t1, t2;

double tt;

int numHebras, tope;

// Comprobacion y extraccion de los argumentos de entrada.

numHebras = 4;

tope = 1000000;

System.*out*.println( "numHebras: " + numHebras );

System.*out*.println( "tope: " + tope );

System.*out*.println( "Creando y arrancando " + numHebras + " hebras." );

t1 = System.*nanoTime*();

MiHebra v[] = new MiHebra[ numHebras ];

AtomicInteger c = new AtomicInteger(0);

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

v[ i ] = new MiHebra( tope, c );

v[ i ].start();

}

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

try {

v[ i ].join();

} catch( InterruptedException ex ) {

ex.printStackTrace();

}

}

t2 = System.*nanoTime*();

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

System.*out*.println( "Total de incrementos: " + c.get() );

System.*out*.println( "Tiempo transcurrido en segs.: " + tt );

}

}

1.7) Completa la siguiente tabla con datos de todas las versiones anteriores en tu ordenador, utilizando hebras 4 y un tope de 1 000 000. Comenta los resultados.

Código Total incrementos

| Código original | **3.695.764** |
| --- | --- |
| Código con volatile | **1.353.735** |
| Código con synchronized | **4.000.000** |
| Código con clases at´omicas | **1.341.118** |

5

| 2 |
| --- |

Se desea imprimir en pantalla aquellos números primos contenidos en un vector.

2.1) Compila y ejecuta el programa anterior, utilizando un 0 como segundo parámetro. En este caso se trabajo con el siguiente vector de números:

long vectorNumeros [ ] = *{*

200000033L , 200000039L , 200000051L , 200000069L ,

200000081L , 200000083L , 200000089L , 200000093L ,

200000107L , 200000117L , 200000123L , 200000131L ,

200000161L , 200000183L , 200000201L , 200000209L ,

4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L

*}* ;

¿Cuáles son los números primos contenidos en el vector?

200000033, 200000039, 200000051, 200000069, 200000081, 200000083, 200000089, 200000093, 200000107, 200000117, 200000123, 200000131, 200000161, 200000183, 200000201, 200000209

**Tiempo secuencial 47.39s**

2.2) Realiza una implementación paralela con **distribución cíclica**, en la que cada hebra procese un conjunto de elementos del vector. Para cada elemento del vector procesado, se mostrará su valor SOLO si es primo.

Incluye la gestión de hebras de esta versión a continuación de la implementación secuencial. Comprueba que los números primos mostrados en la versión paralela coinciden con los de la versión secuencial.

Escribe, a continuación, la parte de tu código que realiza tal tarea: la definición de la clase **MiHebraPrimoDistCiclica** y el código a incluir en el programa principal que permite gestionar los objetos de esta clase.

class MiHebraPrimoDistCiclica extends Thread{

long v[];

int miId;

int numHebras;

MiHebraPrimoDistCiclica(long v[], int miId, int numHebras){

this.miId = miId;

this.v = v;

this.numHebras = numHebras;

}

public void run(){

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

if (esPrimo(v[i]))

System.*out*.println("Encontrado es primo: " +v[i] );

}

}

boolean esPrimo( long num ) {

boolean primo;

if( num < 2 ) {

primo = false;

} else {

primo = true;

long i = 2;

while( ( i < num )&&( primo ) ) {

primo = ( num % i != 0 );

i++;

}

}

return( primo );

}

}

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

public class EjemploMuestraPrimosEnVector {

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

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

public static void main(String args[]) {

int numHebras, vectOpt;

boolean option = true;

long t1, t2;

double ts, tc, tb, td;

// Comprobacion y extraccion de los argumentos de entrada.

numHebras = 4;

vectOpt = 0;

//

// Eleccion del vector de trabajo

//

VectorNumeros vn = new VectorNumeros (option);

long vectorNumeros[] = vn.vector;

//

// Implementacion paralela ciclica.

//

System.*out*.println( "" );

System.*out*.println( "Implementacion paralela ciclica." );

t1 = System.*nanoTime*();

// Gestion de hebras para la implementacion paralela ciclica

// ....

MiHebraPrimoDistCiclica v[] = new MiHebraPrimoDistCiclica[numHebras];

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

v[i] = new MiHebraPrimoDistCiclica(vectorNumeros,i,numHebras);

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

v[i].start();

for (int i = 0; i < v.length; 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: " + ts/tc );

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

9

2.3) Realiza una implementación paralela con **distribución por bloques**, en la que cada hebra procese un conjunto de elementos del vector. Para cada elemento del vector procesado, se mostrará su valor SOLO si es primo.

Incluye la gestión de hebras de esta versión a continuación de la implementación cíclica. Comprueba que los números primos mostrados en la versión paralela coinciden con los de la versión secuencial.

Escribe, a continuación, la parte de tu código que realiza tal tarea: la definición de la clase **MiHebraPrimoDistPorBloques** y el código a incluir en el programa principal que permite gestionar los objetos de esta clase.

----------------- Método run de MiHebraDistPorBloques-----------------

public void run(){

int tam=(v.length +numHebras-1)/numHebras;

int ini=tam\*miId;

int fin =Math.*min*(ini+tam,v.length);

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

if(esPrimo(v[i])){

System.*out*.println("Encontrado es primo: " +v[i]);

}

}

}

----------------- Implementación en EjemploMuestraPrimosEnVector-----------------

System.*out*.println( "" );

System.*out*.println( "Implementacion paralela por bloques." );

t1 = System.*nanoTime*();

MiHebraDistPorBloque vBloque[] = new MiHebraDistPorBloque[numHebras];

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

vBloque[i] = new MiHebraDistPorBloque(vectorNumeros,i,numHebras);

vBloque[i].start();}

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

try {

vBloque[i].join();

}catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

t2 = System.*nanoTime*();

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

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

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

2.4) Realiza una implementación paralela con **distribución dinámica**, utilizando un número entero atómico (AtomicInteger), que apunta a una posición en el vector. Las hebras recibe un único objeto de este tipo, que siempre contiene la primera posición del vector sin procesar. Para ello, las hebras deben realizar de modo atómico, la lectura del valor actual y su incremento. Las hebras finalizar´an cuando el ´ındice sobrepase la dimensi´on del vector. Incluye la gestión de hebras de esta versi´on a continuaci´on de la implementaci´on por bloques. Comprueba que los n´umeros primos mostrados en la versi´on paralela coinciden con los de la versi´on secuencial.

Escribe, a continuaci´on, la parte de tu c´odigo que realiza tal tarea: la definici´on de la clase MiHebraPrimoDistDinamica y el c´odigo a incluir en el programa principal que permite gestionar los objetos de esta clase.

class MiHebraDistDinamica extends Thread {

long v[];

int miId;

AtomicInteger indice;

MiHebraDistDinamica(long v[], int miId, AtomicInteger indice) {

this.miId = miId;

this.v = v;

this.indice = indice;

}

public void run() {

int pos = indice.getAndIncrement();

while (pos < v.length) {

if (esPrimo(v[pos])) {

System.*out*.println("Encontrado es primo: " + v[pos] + " En la hebra: " + miId);

}

pos = indice.getAndIncrement();

}

}

System.*out*.println("Implementacion dinamica");

MiHebraDistDinamica v2[] = new MiHebraDistDinamica[numHebras];

AtomicInteger indice = new AtomicInteger(0);

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

v2[i] = new MiHebraDistDinamica(vectorNumeros,i,indice);

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

v2[i].start();

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

try {

v2[i].join();

}catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

t2 = System.*nanoTime*();

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

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

System.*out*.println( "Incremento paralela dinamica: " + ts/tc );

11

2.5) Completa la siguiente tabla, obteniendo los resultados para 4 hebras en el ordenador del aula y los resultados para 16 hebras en patan. Redondea los tiempos dejando s´olo tres decimales y redondea los incrementos dejando dos decimales.

4 hebras 16 hebras

|  | Tiempo | Incremento | Tiempo | Incremento |
| --- | --- | --- | --- | --- |
| Secuencial | 22.16 | — |  | — |
| Paralela con distribuci´on c´ıclica | 6.54 | 3.66 |  |  |
| Paralela con distribución por bloques | 24.17 | 0.95 |  |  |
| Paralela con distribuci´on din´amica | 6.31 | 3.53 |  |  |

2.6) Justifica los resultados de la tabla anterior.

**Paralela con distribución cíclica:** Al tener 4 o 16 hebras cada una va cogiendo los valores de la siguiente manera (ponemos ejemplo de 4 hebras)

h0→ 0,4,8,12,16,20 h1→ 1,5,9,13,17,21 h2→2,6,10,14,18,22 h3→3,7,11,15,19,23

En este caso el peso de los valores está repartido uniformemente, por lo que el incremento del tiempo llega a ser cercano a 4 (num de hebras)

**Paralela con distribución por bloques:** En este caso cada hebra se encargará de calcular los primos de un bloque (cuyo tamaño será el mismo para todas las hebras). En este caso los valores más pesados los ejecuta todos la hebra 0, por tanto la cantidad de trabajo no está repartida equitativamente.

**Paralela con distribución dinámica:** Es más rápida ya que cada vez que una hebra termina de calcular si el valor es prima puede acceder a la variable atómica para calcular el siguiente valor, de forma que ninguna habrá estará parada mientras qeudan elementos que calcular.

2.7) Eval´ua y compara las tres versiones (secuencial, paralela c´ıclica y paralela por bloques), pero en este caso utilizando 1 como segundo par´ametro, es decir, manejando el vector:

long vectorNumeros [ ] = *{*

200000033L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000039L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000051L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000069L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000081L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000083L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000089L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000093L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000107L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000117L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000123L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000131L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000161L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000183L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000201L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 200000209L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L , 4L

*}* ;

Completa la siguiente tabla, obteniendo los resultados para 4 hebras en el ordenador del aula y los resultados para 16 hebras en patan. Redondea los tiempos dejando s´olo tres decimales y redondea los incrementos dejando dos decimales.

4 hebras 16 hebras

|  | Tiempo | Incremento | Tiempo | Incremento |
| --- | --- | --- | --- | --- |
| Secuencial | 23.26 | — |  | — |
| Paralela con distribución cíclica | 6.22 | 3.78 |  |  |
| Paralela con distribución por bloques | 22.91 | 1.028 |  |  |
| Paralela con distribuci´on dinámica | 6.18 | 3.81 |  |  |

12

2.8) Justifica los resultados de la tabla anterior.

**Paralela con distribución cíclica:** Al tener 4 o 16 hebras cada una va cogiendo los valores de la siguiente manera (ponemos ejemplo de 4 hebras)

h0→ 0,4,8,12,16,20 h1→ 1,5,9,13,17,21 h2→2,6,10,14,18,22 h3→3,7,11,15,19,23

En este caso el peso de los valores está repartido uniformemente, por lo que el incremento del tiempo llega a ser cercano a 4 (num de hebras)

**Paralela con distribución por bloques:** En este caso cada hebra se encargará de calcular los primos de un bloque (cuyo tamaño será el mismo para todas las hebras). En este caso los valores más pesados ya no los ejecuta todos la hebra 0, por tanto la cantidad de trabajo no está tan mal repartida **Paralela con distribución dinámica:** Es más rápida ya que cada vez que una hebra termina de calcular si el valor es prima puede acceder a la variable atómica para calcular el siguiente valor, de forma que ninguna habrá estará parada mientras qeudan elementos que calcular.

2.9) ¿Cuál es la mejor distribución con ambos vectores? Justifica tu respuesta.

La distribución dinámica es la mejor en ambos casos, ya que todas las hebras irán calculando los valores conforme la hebra quede libre (en el peor de los casos solo tendrá que esperar a que otra hebra incremente la variable atómica)

| 3 |
| --- |

Empleando el ordenador del aula, completa la siguiente tabla con datos de todas las versiones desarrolladas en el ejercicio 1, utilizando hebras 4 y un tope de 10 000 000. Redondea los tiempos dejando s´olo tres decimales y comenta los resultados.

Código Total incrementos Tiempo transcurrido (seg.)

| C´odigo original |  |  |
| --- | --- | --- |
| C´odigo con volatile |  |  |
| C´odigo con synchronized |  |  |
| C´odigo con clases at´omicas |  |  |

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