

TRABAJO 3 (IC2)

Ayoze Ruano Alcántara, Enrique Reina Hernández y Fabio Nesta Arteaga Cabrera

Vamos a empezar con el ejercicio de entrenamiento, del cual hay que partir para hacer el trabajo.

Nota: GrupoRAR significa Grupo Reina, Arteaga, Ruano (Nuestros apellidos).

```
[estudiante_ic2@ic2-centos-vm trabajo3]$ make
gcc -c parallel_sum.c -std=gnu99
gcc -c error.c -std=gnu99
gcc -o parallel_sum parallel_sum.o error.o -lpthread -std=gnu99
[estudiante_ic2@ic2-centos-vm trabajo3]$ ./parallel_sum
[ERROR] -----
[ERROR] how many elements do you want to sum? At least must be 8!
[ERROR] In main(): parallel_sum.c:72
[ERROR] -----

usage: ./parallel_sum <numbers_of_elements>

[estudiante_ic2@ic2-centos-vm trabajo3]$
```

Hay que pasarle un número como argumento.

```
[estudiante_ic2@ic2-centos-vm trabajo3]$ ./parallel_sum 10
[000000][main] random vector (10 elements):
[000000][main] --> element 0: 423.510657
[000000][main] --> element 1: -373.401250
[000000][main] --> element 2: 461.532496
[000000][main] --> element 3: 644.732467
[000000][main] --> element 4: 459.406038
[000000][main] --> element 5: 247.014938
[000000][main] --> element 6: -950.608848
[000000][main] --> element 7: -22.634048
[000000][main] --> element 8: -855.463577
[000000][main] --> element 9: -293.201572
[000000][main] thread 0 launched (position: 0, elements: 1)
[000000][thread-0] starting
[000000][thread-0] adding 1 elements starting at position 0:
[000000][thread-0] --> adding element 0: 423.510657
[000000][thread-1] starting
[000000][thread-1] adding 1 elements starting at position 1:
[000000][thread-1] --> adding element 1: -373.401250
[000000][thread-1] obtained sum: -373.401250
[000000][thread-1] finishing
[000000][thread-0] obtained sum: 423.510657
[000000][thread-0] finishing
[000000][main] thread 1 launched (position: 1, elements: 1)
[000000][main] thread 2 launched (position: 2, elements: 1)
[000000][thread-2] starting
[000000][thread-2] adding 1 elements starting at position 2:
[000000][thread-2] --> adding element 2: 461.532496
[000000][thread-2] obtained sum: 461.532496
[000000][thread-2] finishing
[000000][thread-3] starting
[000000][thread-3] adding 1 elements starting at position 3:
[000000][main] thread 3 launched (position: 3, elements: 1)
[000000][thread-3] --> adding element 3: 644.732467
[000000][thread-3] obtained sum: 644.732467
```

Vamos a preguntarle a Chat GPT que nos explique la funcionalidad del código.



ChatGPT

Este código en C implementa un programa que calcula la suma de un vector de números de forma secuencial y también de forma paralela utilizando múltiples hilos. Aquí está una explicación detallada de su funcionalidad:

Inclusión de bibliotecas

El programa incluye varias bibliotecas estándar de C y otras específicas para manejar hilos (`pthread.h`) y errores (`error.h`).

Declaración de constantes y variables globales

- `MAX RAND VALUE`: Define el valor máximo para los elementos aleatorios del vector.
- `ZERO DIFF`: Define una pequeña diferencia para comparaciones de punto flotante.
- `initial_time`: Variable global para registrar el tiempo inicial.

Declaración de estructuras

- `thread_ctx`: Una estructura que almacena información necesaria para cada hilo, como un identificador, un puntero al vector de números, la posición de inicio en el vector, el número de elementos a sumar y la suma acumulada.

Función `time_in_millis`

Esta función utiliza `gettimeofday` para obtener el tiempo actual en milisegundos desde el inicio del programa.

Función `thread`

Es la función que ejecutará cada hilo. Se encarga de sumar los elementos del vector asignados a ese hilo.

Función `main`

- Inicializa variables locales y verifica los argumentos de línea de comandos.
- Genera un vector de números aleatorios y lo muestra en la salida estándar.
- Crea múltiples hilos, cada uno de los cuales sumará una parte del vector.
- Espera a que todos los hilos terminen.
- Calcula la suma total tanto de forma secuencial como paralela, y compara los resultados.

Explicación del proceso de suma paralela

- El vector se divide en partes iguales, cada una asignada a un hilo.
- Cada hilo suma los elementos de su parte asignada.
- Una vez que todos los hilos terminan, se suman sus resultados individuales para obtener la suma total.

Salida del programa

- Muestra información sobre la ejecución de cada hilo.
- Muestra los resultados de las sumas paralela y secuencial, y si son iguales o no.

Manejo de errores

- El código maneja posibles errores, como la creación de hilos fallida o errores al obtener el tiempo.

Realice las siguientes modificaciones:

1. Modifique **parallel_sum.c** de manera que la división en trozos a sumar por cada hilo, sea lo más equilibrada posible. Si observa el código, en el ejemplo el último hilo se encarga de sumar más elementos, en el caso de que el número de elementos a sumar no sea divisible por el número de hilos.
2. Modifique el ejemplo de forma que el hilo **main** también realice trabajo productivo, responsabilizándose de uno de los trozos de datos a sumar.

```
[000001][main] waiting for thread 3
[000001][main] thread 3 joined
[000001][main] waiting for thread 4
[000000][thread-7] starting
[000001][thread-7] adding 8 elements starting at position 7:
[000001][thread-7] --> adding element 7: 447.306898
[000001][thread-7] --> adding element 8: -758.134914
[000001][thread-6] obtained sum: -122.456746
[000001][thread-6] finishing
[000001][main] thread 4 joined
[000001][main] waiting for thread 5
[000001][thread-7] --> adding element 9: -854.090729
[000001][thread-7] --> adding element 10: -719.728043
[000001][thread-7] --> adding element 11: -208.091544
[000001][thread-7] --> adding element 12: 484.689575
[000001][thread-7] --> adding element 13: 274.203604
[000001][thread-7] --> adding element 14: 216.863471
[000001][thread-7] obtained sum: -1116.981681
```

Al poner un número no divisible entre 8 (números de hilos disponibles), le asigna un elemento a cada hilo, menos al último, al que se le asignan todos los elementos restantes.

```
[000000][main] thread 6 launched (position: 12, elements: 2)
[000000][thread-5] --> adding element 11: -100.992865
[000001][thread-7] starting
[000001][thread-7] adding 2 elements starting at position 14:
[000001][thread-7] --> adding element 14: 128.469308
[000001][thread-7] --> adding element 15: -467.838697
[000001][thread-7] obtained sum: -339.369389
[000001][thread-7] finishing
[000001][thread-5] obtained sum: -1094.666316
[000001][thread-5] finishing
[000001][main] thread 7 launched (position: 14, elements: 2)
[000001][main] waiting for thread 0
```

Al poner un número divisible entre 8, se le asigna a cada grupo el mismo número de elementos.

```
pthread_t the_threads[n_threads];
thread_ctx thread_ctxs[n_threads];
int thread_elems=n_elems/n_threads; // tenemos que modificar thread_elems, para asignar cierto
// numero de elementos a cada hilo
```

Operador condicional

Es un operador ternario, es decir, tiene tres operandos.

Su funcionamiento es el siguiente: para una ejecución $c ? e_1 : e_2$ siendo c , e_1 y e_2 tres expresiones,

- Se evalúa c
 - Si el resultado es *cierto* (es decir, distinto de cero)
 - se evalúa e_1 y éste sera el resultado.
 - Si no (el resultado es *falso* o cero)
 - se evalúa e_2 y éste sera el resultado.

Ejemplos. Para $x = 4$, $y = 2$, $z = 7$:

- La expresión $x >= 5 ? 1 : 0$ se evalúa a 1
- La expresión $x >= 5 ? y : z$ se evalúa a 7
- La expresión $x <= 5 ? y : z$ se evalúa a 2
- La expresión $x ? z+y : z-y$ se evalúa a 9
- La expresión $x >= 0 ? \text{sqrt}(x) : 0$ se evalúa a 2.0 (*sqrt*: raíz cuadrada, prototipo en `math.h`)

DdF	C	Descripción
: ?	: ?	Operador condicional $c ? e_1 : e_2$

Operadores aritméticos

Cabe destacar que se diferencian la división entera y la división real, según los operandos.

DdF	C	Descripción
+	+	Suma, adición
-	-	Resta, sustracción
.	*	Multiplicación, producto
<u>Div</u>	/	Cociente división entera
<u>Mod</u>	%	Resto división entera
/	/	División

Buscamos para qué servía el `?` y como hacer el módulo.

$\wedge \ 0$	t_0	$0 \ (1+1)$	$0 = 0$
$C: 1$	t_1	$1 \ (1+1)$	$2 = 2$
$R: 2 = 5$	t_2	$2 \ (1)+5$	$2+2 = 4$
	t_3	$3 \ 1+5$	$3+2 = 5$
	t_4	$4 \ 1+5$	$4+2 = 6$
	t_5	$5 \ 1+5$	$5+2 = 7$
	\vdots	\vdots	

Esto es cómo diseñamos el comportamiento que va a tomar el apartado position del struct de cada hilo. Para 10 elementos, tenemos un Cociente de 1 y un Resto de 2.

Implementamos ese algoritmo matemático en C, modificando el código base; además el apartado elems de cada struct va a tomar el cociente+1, en caso de que se encuentre en el rango de la longitud del resto, que estamos utilizando como contador realmente.

```
pthread_t the_threads[n_threads];
thread_ctx thread_ctxs[n_threads];
int thread_elems=n_elems/n_threads; // Cociente de los elementos creados entre el numero
// de hilos existentes
int resto = n_elems % n_threads; // cantidad restante de elementos a asignar en caso de
// division no exacta (va a actuar como counter)
int sobrante = n_elems % n_threads; // va a conservar el valor inicial del resto

for(int i=0; i<n_threads; i++)
{
    thread_ctxs[i].id=i;
    thread_ctxs[i].vector=v;
    thread_ctxs[i].position=(resto > 0)?
        i*(thread_elems+1) : i*(thread_elems)+sobrante;

    thread_ctxs[i].elems=(resto > 0)?
        thread_elems+1 : thread_elems;

    resto--;
}
```

Ponemos un output para 10 elementos, y otro para 21.



Archivo Editar Ver Buscar Terminal Ayuda

```
[estudiante_ic2@ic2-centos-vm trabajo3]$ ./parallel_sum 10
[000000][main] random vector (10 elements):
[000000][main] --> element 0: -235.898391
[000000][main] --> element 1: -126.227033
[000000][main] --> element 2: 738.617816
[000000][main] --> element 3: -258.520200
[000000][main] --> element 4: -501.718539
[000000][main] --> element 5: -309.669284
[000000][main] --> element 6: -164.735106
[000000][main] --> element 7: 666.299759
[000000][main] --> element 8: -885.239310
[000000][main] --> element 9: -113.343000
[000000][main] thread 0 launched (position: 0, elements: 2)
[000000][main] thread 1 launched (position: 2, elements: 2)
[000000][main] thread 2 launched (position: 4, elements: 1)
[000000][main] thread 3 launched (position: 5, elements: 1)
[000000][main] thread 4 launched (position: 6, elements: 1)
[000000][main] thread 5 launched (position: 7, elements: 1)
[000000][main] thread 6 launched (position: 8, elements: 1)
[000000][main] thread 7 launched (position: 9, elements: 1)
[000000][main] waiting for thread 0
[000000][thread-1] starting
[000000][thread-1] adding 2 elements starting at position 2:
[000000][thread-1] --> adding element 2: 738.617816
[000000][thread-1] --> adding element 3: -258.520200
[000000][thread-1] obtained sum: 480.097616
[000000][thread-1] finishing
[000000][thread-3] starting
[000000][thread-3] adding 1 elements starting at position 5:
[000000][thread-3] --> adding element 5: -309.669284
[000000][thread-3] obtained sum: -309.669284
[000000][thread-3] finishing
[000000][thread-0] starting
[000000][thread-0] adding 2 elements starting at position 0:
[000000][thread-0] --> adding element 0: -235.898391
[000000][thread-0] --> adding element 1: -126.227033
[000000][thread-0] obtained sum: -362.125424
[000000][thread-0] finishing
[000000][thread-2] starting
[000000][thread-2] adding 1 elements starting at position 4:
[000000][thread-2] --> adding element 4: -501.718539
[000000][thread-2] obtained sum: -501.718539
[000000][thread-2] finishing
[000000][thread-4] starting
[000000][thread-4] adding 1 elements starting at position 6:
```

```
[000000][thread-4] --> adding element 6: -164.735106
[000000][thread-4] obtained sum: -164.735106
[000000][thread-4] finishing
[000000][main] thread 0 joined
[000000][main] waiting for thread 1
[000000][main] thread 1 joined
[000000][main] waiting for thread 2
[000000][main] thread 2 joined
[000000][main] waiting for thread 3
[000000][main] thread 3 joined
[000000][main] waiting for thread 4
[000000][main] thread 4 joined
[000000][main] waiting for thread 5
[000000][thread-5] starting
[000000][thread-5] adding 1 elements starting at position 7:
[000000][thread-5] --> adding element 7: 666.299759
[000000][thread-5] obtained sum: 666.299759
[000000][thread-5] finishing
[000000][main] thread 5 joined
[000000][main] waiting for thread 6
[000000][thread-6] starting
[000001][thread-6] adding 1 elements starting at position 8:
[000001][thread-6] --> adding element 8: -885.239310
[000001][thread-6] obtained sum: -885.239310
[000001][thread-6] finishing
[000001][main] thread 6 joined
[000001][main] waiting for thread 7
[000001][thread-7] starting
[000001][thread-7] adding 1 elements starting at position 9:
[000001][thread-7] --> adding element 9: -113.343000
[000001][thread-7] obtained sum: -113.343000
[000001][thread-7] finishing
[000001][main] thread 7 joined
[000001][main] results:
[000001][main] --> parallel sum: -1190.433287
[000001][main] --> sequential sum: -1190.433287
[000001][main] --> are equal, RIGHT!
```



Archivo Editar Ver Buscar Terminal Ayuda

```
[estudiante_ic2@ic2-centos-vm trabajo3]$ ./parallel_sum 21
[000000][main] random vector (21 elements):
[000000][main] --> element 0: 821.099657
[000000][main] --> element 1: 592.016308
[000000][main] --> element 2: -140.665239
[000000][main] --> element 3: 543.561736
[000000][main] --> element 4: 834.717099
[000000][main] --> element 5: -115.182239
[000000][main] --> element 6: 604.155161
[000000][main] --> element 7: -247.203242
[000000][main] --> element 8: -995.787950
[000000][main] --> element 9: -402.736725
[000000][main] --> element 10: -635.688139
[000000][main] --> element 11: 193.165354
[000000][main] --> element 12: 523.845105
[000000][main] --> element 13: -935.281212
[000000][main] --> element 14: 46.552369
[000000][main] --> element 15: -885.500220
[000000][main] --> element 16: 159.733671
[000000][main] --> element 17: 500.110031
[000000][main] --> element 18: -74.463320
[000000][main] --> element 19: 999.299341
[000000][main] --> element 20: -464.070350
[000000][main] thread 0 launched (position: 0, elements: 3)
[000001][thread-0] starting
[000001][thread-0] adding 3 elements starting at position 0:
[000001][thread-0] --> adding element 0: 821.099657
[000001][thread-0] --> adding element 1: 592.016308
[000001][thread-0] --> adding element 2: -140.665239
[000001][thread-0] obtained sum: 1272.450727
[000001][thread-0] finishing
[000001][main] thread 1 launched (position: 3, elements: 3)
[000001][thread-1] starting
[000001][thread-1] adding 3 elements starting at position 3:
[000001][thread-1] --> adding element 3: 543.561736
[000001][thread-1] --> adding element 4: 834.717099
[000001][thread-1] --> adding element 5: -115.182239
[000001][thread-1] obtained sum: 1263.096596
[000001][thread-2] starting
[000001][thread-2] adding 3 elements starting at position 6:
[000001][thread-2] --> adding element 6: 604.155161
[000001][thread-2] --> adding element 7: -247.203242
[000001][thread-2] --> adding element 8: -995.787950
[000001][thread-2] obtained sum: -638.836031
[000001][thread-2] finishing
[000001][thread-1] finishing
```



```
estudiante_ic2@ic2-centos-vm:~/grupoRAR/trabajo3
Archivo Editar Ver Buscar Terminal Ayuda
[000001][thread-1] finishing
[000001][main] thread 2 launched (position: 6, elements: 3)
[000002][main] thread 3 launched (position: 9, elements: 3)
[000002][main] thread 4 launched (position: 12, elements: 3)
[000002][main] thread 5 launched (position: 15, elements: 2)
[000002][main] thread 6 launched (position: 17, elements: 2)
[000002][main] thread 7 launched (position: 19, elements: 2)
[000002][main] waiting for thread 0
[000002][main] thread 0 joined
[000002][main] waiting for thread 1
[000002][main] thread 1 joined
[000002][main] waiting for thread 2
[000002][main] thread 2 joined
[000002][main] waiting for thread 3
[000002][thread-5] starting
[000002][thread-5] adding 2 elements starting at position 15:
[000002][thread-5] --> adding element 15: -885.500220
[000002][thread-5] --> adding element 16: 159.733671
[000002][thread-5] obtained sum: -725.766548
[000002][thread-5] finishing
[000002][thread-7] starting
[000002][thread-7] adding 2 elements starting at position 19:
[000002][thread-7] --> adding element 19: 999.299341
[000002][thread-7] --> adding element 20: -464.070350
[000002][thread-7] obtained sum: 535.228991
[000002][thread-7] finishing
[000002][thread-3] starting
[000002][thread-3] adding 3 elements starting at position 9:
[000002][thread-3] --> adding element 9: -402.736725
[000002][thread-3] --> adding element 10: -635.688139
[000002][thread-3] --> adding element 11: 193.165354
[000002][thread-3] obtained sum: -845.259510
[000002][thread-3] finishing
[000002][thread-4] starting
[000002][thread-4] adding 3 elements starting at position 12:
[000002][thread-4] --> adding element 12: 523.845105
[000002][thread-4] --> adding element 13: -935.281212
[000002][thread-4] --> adding element 14: 46.552369
[000002][thread-4] obtained sum: -364.883738
[000002][thread-4] finishing
[000003][thread-6] starting
[000003][thread-6] adding 2 elements starting at position 17:
[000003][thread-6] --> adding element 17: 500.110031
[000003][thread-6] --> adding element 18: -74.463320
[000003][thread-6] obtained sum: 425.646711
[000003][thread-6] finishing
```

```
[000003][main] thread 3 joined
[000003][main] waiting for thread 4
[000003][main] thread 4 joined
[000003][main] waiting for thread 5
[000003][main] thread 5 joined
[000003][main] waiting for thread 6
[000003][main] thread 6 joined
[000003][main] waiting for thread 7
[000003][main] thread 7 joined
[000003][main] results:
[000003][main] --> parallel sum: 921.677198
[000003][main] --> sequential sum: 921.677198
[000003][main] --> are equal, RIGHT!
[estudiante_ic2@ic2-centos-vm trabajo3]$
```

2. Modifique el ejemplo de forma que el hilo **main** también realice trabajo productivo, responsabilizándose de uno de los trozos de datos a sumar.

```
pthread_t the_threads[n_threads];
thread_ctx thread_ctxs[n_threads];
int thread_elems=n_elems/(n_threads+1); // Cociente de los elementos creados entre el numero
// de hilos existentes
int resto = n_elems % (n_threads+1); // cantidad restante de elementos a asignar en caso de
// division no exacta (va a actuar como counter)
int sobrante = n_elems % (n_threads+1); // va a conservar el valor inicial del resto

for(int i=0; i<n_threads; i++)
{
    thread_ctxs[i].id=i;
    thread_ctxs[i].vector=v;
    thread_ctxs[i].position=(resto > 0)?
        i*(thread_elems+1) : i*(thread_elems)+sobrante;

    thread_ctxs[i].elems=(resto > 0)?
        thread_elems+1 : thread_elems;

    resto--;
}
```

Cambiamos el código para que se le deje un fragmento de datos al main, de modo que la división la hacemos en 9, ya que tenemos 8 hilos y el main actúa como noveno hilo.

```
int main_position = (resto > 0)?
    thread_ctxs[7].position+thread_elems+1 : thread_ctxs[7].position+thread_elems;

double sum_main = 0;

for (int i = main_position; i < n_elems; i++){
    printf("valor: %f %d\n", v[i], i);
    sum_main += v[i];
}

printf("[%06ld][%s] results:\n",time_in_millis()-initial_time,__func__);
double parallel_sum=0;
for(int i=0; i<n_threads; i++) parallel_sum+=thread_ctxs[i].sum;

printf("Suma de los hilos (sin el hilo main): %f\n", parallel_sum);
parallel_sum+= sum_main;
```

Abajo le damos instrucciones al main para realizar trabajo productivo, básicamente cogiendo el último trozo que queda tras la asignación en los otros hilos. Vamos a ver el output, y luego lo pulimos si hace falta.

OUTPUT PARA 19 !

```
estudiante_ic2@ic2-centos-vm:~/grupoRAR/trabajo3
Archivo Editar Ver Buscar Terminal Ayuda
[estudiante_ic2@ic2-centos-vm trabajo3]$ ./parallel_sum 19
[000000][main] random vector (19 elements):
[000000][main] --> element 0: 19.921823
[000000][main] --> element 1: -475.043382
[000000][main] --> element 2: 748.378121
[000000][main] --> element 3: 230.225394
[000000][main] --> element 4: -665.020521
[000000][main] --> element 5: -440.261182
[000000][main] --> element 6: -590.158343
[000000][main] --> element 7: -263.453520
[000000][main] --> element 8: 214.353545
[000000][main] --> element 9: 156.258986
[000000][main] --> element 10: -565.609912
[000000][main] --> element 11: -175.648362
[000000][main] --> element 12: -634.599623
[000000][main] --> element 13: 586.572070
[000000][main] --> element 14: -743.601536
[000000][main] --> element 15: -639.627254
[000000][main] --> element 16: -815.362845
[000000][main] --> element 17: 775.946443
[000000][main] --> element 18: -310.159706
[000000][main] thread 0 launched (position: 0, elements: 3)
[000000][main] thread 1 launched (position: 3, elements: 2)
[000000][main] thread 2 launched (position: 5, elements: 2)
[000000][main] thread 3 launched (position: 7, elements: 2)
[000000][main] thread 4 launched (position: 9, elements: 2)
[000000][main] thread 5 launched (position: 11, elements: 2)
[000000][main] thread 6 launched (position: 13, elements: 2)
[000000][main] thread 7 launched (position: 15, elements: 2)
[000000][main] waiting for thread 0
[000000][thread-0] starting
[000000][thread-0] adding 3 elements starting at position 0:
[000000][thread-0] --> adding element 0: 19.921823
[000000][thread-0] --> adding element 1: -475.043382
[000000][thread-0] --> adding element 2: 748.378121
[000000][thread-0] obtained sum: 293.256562
[000000][thread-0] finishing
[000000][thread-1] starting
[000000][thread-1] adding 2 elements starting at position 3:
[000000][thread-1] --> adding element 3: 230.225394
[000000][thread-1] --> adding element 4: -665.020521
[000000][thread-1] obtained sum: -434.795127
[000000][thread-1] finishing
[000003][thread-2] starting
[000003][thread-2] adding 2 elements starting at position 5:
[000003][thread-2] --> adding element 5: -440.261182
```

```
estudiante_ic2@ic2-centos-vm:~/grupoRAR/trabajo3
Archivo Editar Ver Buscar Terminal Ayuda
[000003][thread-2] --> adding element 5: -440.261182
[000003][thread-2] --> adding element 6: -590.158343
[000003][thread-2] obtained sum: -1030.419525
[000003][thread-2] finishing
[000003][thread-4] starting
[000003][thread-4] adding 2 elements starting at position 9:
[000003][thread-4] --> adding element 9: 156.258986
[000003][thread-4] --> adding element 10: -565.609912
[000003][thread-4] obtained sum: -409.350926
[000003][thread-4] finishing
[000004][main] thread 0 joined
[000004][main] waiting for thread 1
[000004][thread-5] starting
[000004][thread-5] adding 2 elements starting at position 11:
[000004][thread-5] --> adding element 11: -175.648362
[000004][thread-5] --> adding element 12: -634.599623
[000004][thread-5] obtained sum: -810.247985
[000004][thread-5] finishing
[000004][main] thread 1 joined
[000004][main] waiting for thread 2
[000004][main] thread 2 joined
[000004][main] waiting for thread 3
[000004][thread-7] starting
[000004][thread-7] adding 2 elements starting at position 15:
[000004][thread-7] --> adding element 15: -639.627254
[000004][thread-7] --> adding element 16: -815.362845
[000004][thread-7] obtained sum: -1454.990100
[000004][thread-7] finishing
[000005][thread-3] starting
[000005][thread-3] adding 2 elements starting at position 7:
[000005][thread-3] --> adding element 7: -263.453520
[000005][thread-3] --> adding element 8: 214.353545
[000005][thread-3] obtained sum: -49.099975
[000005][thread-3] finishing
[000005][main] thread 3 joined
[000005][main] waiting for thread 4
[000005][main] thread 4 joined
[000005][main] waiting for thread 5
[000005][main] thread 5 joined
[000005][main] waiting for thread 6
[000004][thread-6] starting
[000005][thread-6] adding 2 elements starting at position 13:
[000005][thread-6] --> adding element 13: 586.572070
[000005][thread-6] --> adding element 14: -743.601536
[000005][thread-6] obtained sum: -157.029466
[000005][thread-6] finishing
```

```
[000005][thread-6] finishing
[000005][main] thread 6 joined
[000005][main] waiting for thread 7
[000006][main] thread 7 joined
valor: 775.946443 17
valor: -310.159706 18
[000006][main] results:
Suma de los hilos (sin el hilo main): -4052.676542
[000006][main] --> parallel sum: -3586.889806
[000006][main] --> sequential sum: -3586.889806
[000006][main] --> are equal, RIGHT!
[estudiante_ic2@ic2-centos-vm trabajo3]$
```

Arriba se muestra cuántos datos coge cada hilo (y cuáles en específico), y abajo los datos de los que se encarga el main.

Antes de llegar a este resultado, **el cual es correcto**, habíamos cometido el siguiente error:

```
pthread_t the_threads[n_threads];
thread_ctx thread_ctxs[n_threads];
int thread_elems=n_elems/n_threads+1; // Cociente de los elementos creados entre el numero
// de hilos existentes
int resto = n_elems % n_threads+1; // cantidad restante de elementos a asignar en caso de
// division no exacta (va a actuar como counter)
int sobrante = n_elems % n_threads+1; // va a conservar el valor inicial del resto
```

```
[000001][thread-5] adding 3 elements starting at position 19:
[000001][thread-5] --> adding element 19: 0.000000
[000001][thread-5] --> adding element 20: 0.000000
[000001][thread-5] --> adding element 21: 0.000000
[000001][thread-5] obtained sum: 0.000000
[000001][thread-5] finishing
[000001][thread-7] starting
[000001][thread-7] adding 3 elements starting at position 25:
[000001][thread-7] --> adding element 25: 0.000000
[000001][thread-7] --> adding element 26: 0.000000
[000001][thread-7] --> adding element 27: 0.000000
[000002][thread-7] obtained sum: 0.000000
[000002][thread-7] finishing
[000002][thread-4] starting
[000002][thread-4] adding 3 elements starting at position 16:
[000002][thread-4] --> adding element 16: 967.925889
[000002][thread-4] --> adding element 17: -505.666164
[000002][thread-4] --> adding element 18: -713.294516
[000002][thread-4] obtained sum: -251.034790
[000002][thread-4] finishing
[000002][main] thread 4 joined
[000002][main] waiting for thread 5
[000002][main] thread 5 joined
[000002][main] waiting for thread 6
[000002][thread-6] starting
[000002][thread-6] adding 3 elements starting at position 22:
[000002][thread-6] --> adding element 22: 0.000000
[000002][thread-6] --> adding element 23: 0.000000
[000002][thread-6] --> adding element 24: 0.000000
[000002][thread-6] obtained sum: 0.000000
[000002][thread-6] finishing
[000002][main] thread 6 joined
[000002][main] waiting for thread 7
[000002][main] thread 7 joined
[000002][main] results:
Suma de los hilos (sin el hilo main): -2441.310853
[000002][main] --> parallel sum: -2441.310853
[000002][main] --> sequential sum: -2441.310853
[000002][main] --> are equal, RIGHT!
```


No pusimos paréntesis en el divisor y en los módulos.

Da RIGHT porque los hilos se encargan de hacer la suma, pero como le mandamos una cantidad mayor a los elementos introducidos por consola (en este caso 19, el programa coge $28 + 0$ en el main) debido a las operaciones matemáticas erróneas.

Ahora pasamos a realizar el trabajo como tal.

6 Ejercicio entregable: Producto escalar

A partir del ejercicio de entrenamiento de suma paralela, cuyo código **parallel_sum.c** está disponible en el Campus Virtual de la asignatura, modifique el ejemplo de manera que implemente el producto escalar de dos vectores.

6.1 Implementación básica

A partir del ejercicio de entrenamiento **parallell_sum** adáptelo para que lleve a cabo el producto escalar de dos vectores almacenados en dos vectores de dimensión **n**, **guardando el resultado en una variable global (int R)**. Considere que ambos vectores están contenidos en dos arrays de tipo double con el mismo tamaño, y que se generan aleatoriamente.

Al igual que en el ejemplo **parallel_sum** el programa resultante debe realizar el producto escalar de ambos vectores de manera parcial por cada uno de los hilos lanzados desde el programa principal, **pero ir sumando el resultado a la variable global R**. E igualmente, que en el ejemplo **parallel_sum**, debe mostrar el resultado de realizar el cálculo en el hilo principal (guardo en una variable local **R2**), y el resultante de los productos escalares parciales realizados por cada hilo antes de finalizar su ejecución. Evidentemente ambos valores deben coincidir. Po lo que hay que verificar con más de 1000 iteraciones que el resultado es siempre el mismo, es decir que **no ocurre que (R2!=R)**. Para ello habrá que tener en cuenta lo realizado en clase y ver el ejemplo de Productores-consumidores.

Recuerde que dados dos vectores $a, b \in \mathbb{R}$ tales que $a = (a_0, a_1, \dots, a_{n-1})$ y $b = (b_0, b_1, \dots, b_{n-1}) \forall n \in \mathbb{N}$ su producto escalar se define como:

$$a \cdot b = \sum_{i=0}^{n-1} a_i \cdot b_i$$

Debe denominar al programa **parallell_dot.c**

```
Abrir  parallel_sum.c
~/grupoRAR/trabajo3

parallel_sum.c x

#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <sys/time.h>
#include <math.h>
#include <pthread.h>

#include "error.h"

#define MAX RAND_VALUE 1000
#define ZERO_DIFF ((double) 1e-9)

unsigned long initial_time;

// variable global
int R = 0;

typedef struct
{
    unsigned int id;
    double* vector1;
    double* vector2;
    int position;
    int elems;
    double sum;
} thread_ctx;

void* thread(void* thread_arg) // se modifica para adaptarse al producto escalar
{
    thread_ctx* ctx=(thread_ctx*) thread_arg;

    printf("[%06ld][%-u] starting\n",time_in_millis()-initial_time, __func__,ctx->id);
    //printf("[%06ld][%-u] adding %d elements starting at position %d:\n",time_in_millis()-initial_time,__func__,ctx->id,ctx->elems,ctx->position);

    ctx->sum=0;
    for(int i=ctx->position; i<ctx->position+ctx->elems; i++)
    {
        //printf("[%06ld][%-u] --> adding element %d: %f\n",time_in_millis()-initial_time,__func__,ctx->id,i,ctx->vector[i]);
        ctx->sum +=ctx->vector1[i]*ctx->vector2[i];
    }

    printf("[%06ld][%-u] obtained sum: %f\n",time_in_millis()-initial_time, __func__,ctx->id,ctx->sum);
    printf("[%06ld][%-u] finishing\n",time_in_millis()-initial_time, __func__,ctx->id);
    return thread_arg;
}

srand(time(NULL));
double v1[n_elems]; // VECTOR 1
double v2[n_elems]; // VECTOR 2

/*printf(
    "[%06ld][%-s] random vector (%d elements):\n",
    time_in_millis()-initial_time,
    __func__,
    n_elems
); */

// CREAMOS LOS VECTORES CON VALORES ALEATORIOS
printf("Valores del VECTOR 1\n");
for(int i=0; i<n_elems; i++)
{
    v1[i]=((double)(2*MAX_RAND_VALUE)*rand())/((double) RAND_MAX)-MAX_RAND_VALUE;
    printf(
        "[%06ld][%-s] --> element %d: %f\n",
        time_in_millis()-initial_time,
        __func__,i,v1[i]
    );
}

printf("\n");

printf("Valores del VECTOR 2\n");
for(int i=0; i<n_elems; i++)
{
    v2[i]=((double)(2*MAX_RAND_VALUE)*rand())/((double) RAND_MAX)-MAX_RAND_VALUE;
    printf(
        "[%06ld][%-s] --> element %d: %f\n",
        time_in_millis()-initial_time,
        __func__,i,v2[i]
    );
}
```

```

pthread_t the_threads[n_threads];
thread_ctx thread_ctxs[n_threads];
int thread_elems=n_elems/(n_threads+1); // Cociente de los elementos creados entre el numero
// de hilos existentes
int resto = n_elems % (n_threads+1); // cantidad restante de elementos a asignar en caso de
// division no exacta (va a actuar como counter)
int sobrante = n_elems % (n_threads+1); // va a conservar el valor inicial del resto

for(int i=0; i<n_threads; i++)
{
    thread_ctxs[i].id=i;
    thread_ctxs[i].vector1=v1;
    thread_ctxs[i].vector2=v2;
    thread_ctxs[i].position=(resto > 0)?
        i*(thread_elems+1) : i*(thread_elems)+sobrante;

    thread_ctxs[i].elems=(resto > 0)?
        thread_elems+1 : thread_elems;

    resto--;

    int main_position = (resto > 0)?
        thread_ctxs[7].position+thread_elems+1 : thread_ctxs[7].position+thread_elems;

double sum_main = 0;

    for (int i = main_position; i < n_elems; i++){
        //printf("valor: %f %d\n", v[i], i);
        sum_main += v1[i]*v2[i];
    }

    printf("[%06ld][%s] results:\n",time_in_millis()-initial_time,__func__);
    //double parallel_sum=0;

    for(int i=0; i<n_threads; i++) R +=thread_ctxs[i].sum;

    printf("Suma de los hilos (sin el hilo main): %f\n", R);
    R += sum_main;

    printf("[%06ld][%s] --> parallel sum: %f\n",time_in_millis()-initial_time,__func__, R);


//double sequential_sum=0;
double R2 = 0;
for(int i=0; i<n_elems; i++) R2 += v1[i]*v2[i];
printf("[%06ld][%s] --> sequential sum: %f\n",time_in_millis()-initial_time,__func__, R2);
printf(
    "[%06ld][%s] --> %s\n",
    time_in_millis()-initial_time,
    __func__,
    (fabs(R-R2)<ZERO_DIFF? "are equal, RIGHT!" : "not equal, WRONG!!")
);

return EXIT_SUCCESS;
}

```

Nuestro primer intento. A continuación modificamos el Makefile y el nombre del archivo.

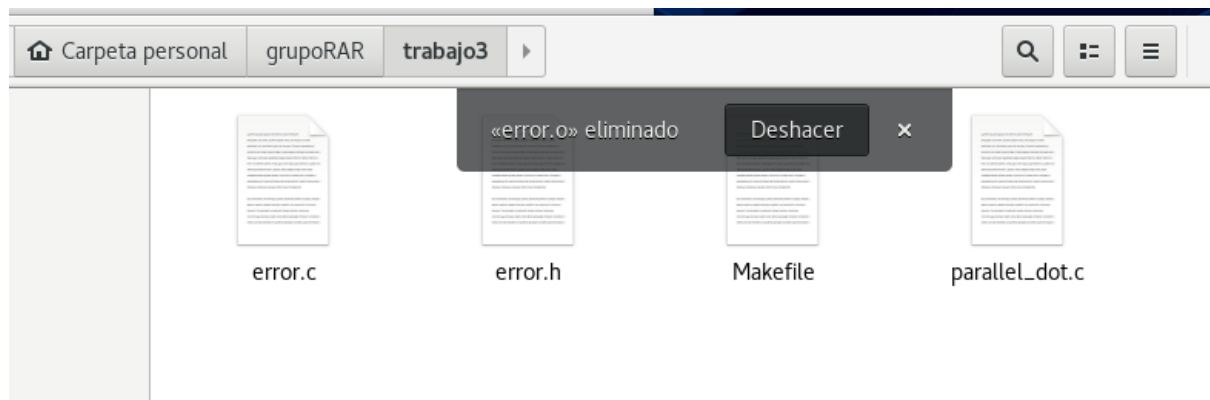
Abrir  Makefile
~/grupoRAR/trabajo3

```
parallel_dot: parallel_dot.o error.o
gcc -o $@ $^ -lpthread -std=gnu99

parallel_dot.o: parallel_dot.c
gcc -c $^ -std=gnu99

error.o: error.c
gcc -c $^ -std=gnu99

clean:
rm *.o parallel_dot
```



```
[estudiante_ic2@ic2-centos-vm trabajo3]$ ./parallel_dot 10
Valores del VECTOR 1
[000000][main] --> element 0: 629.894323
[000000][main] --> element 1: 146.235968
[000000][main] --> element 2: 596.531599
[000000][main] --> element 3: 100.151321
[000000][main] --> element 4: -129.390642
[000000][main] --> element 5: -992.199666
[000000][main] --> element 6: -407.619484
[000000][main] --> element 7: 920.854363
[000000][main] --> element 8: 394.266303
[000000][main] --> element 9: -843.477629

Valores del VECTOR 2
[000000][main] --> element 0: 18.443193
[000000][main] --> element 1: 516.801187
[000000][main] --> element 2: 91.247875
[000000][main] --> element 3: -361.230609
[000000][main] --> element 4: 137.138719
[000000][main] --> element 5: 508.559000
[000000][main] --> element 6: -790.985649
[000000][main] --> element 7: -422.390482
[000000][main] --> element 8: -714.028383
[000000][main] --> element 9: -663.322192
[000000][main] thread 0 launched (position: 0, elements: 2)
[000000][main] thread 1 launched (position: 2, elements: 1)
[000000][main] thread 2 launched (position: 3, elements: 1)
[000000][main] thread 3 launched (position: 4, elements: 1)
[000000][main] thread 4 launched (position: 5, elements: 1)
[000000][main] thread 5 launched (position: 6, elements: 1)
[000000][main] thread 6 launched (position: 7, elements: 1)
[000000][main] thread 7 launched (position: 8, elements: 1)
```

Al poner 10, el cociente entre 9 (contando el main) es 1 y el resto es 1. A todos los hilos (main inclusive) se le asigna UNA PAREJA DE ELEMENTOS (por ejemplo, en el thread 1, va a hacer el producto escalar parcial de $v1[2]$ y $v2[2]$). Cabe destacar que, al haber un resto de 1, en este caso el thread 0 coge DOS PAREJAS DE ELEMENTOS (se encarga de $v1[0]$ y $v2[0]$; y de $v1[1]$ y $v2[1]$). Mirando la captura de arriba, al menos sabemos que los vectores se generan correctamente y que las parejas de valores también se distribuyen equilibradamente.

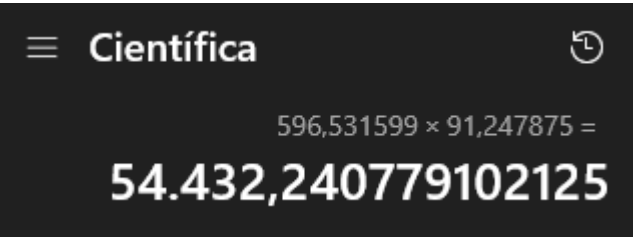
```
[000000][main] waiting for thread 0
[000000][thread-1] starting
[000000][thread-1] obtained sum: 54432.240639
[000000][thread-1] finishing
[000000][thread-3] starting
[000000][thread-3] obtained sum: -17744.466880
[000000][thread-3] finishing
[000000][thread-5] starting
[000001][thread-5] obtained sum: 322421.162577
[000001][thread-5] finishing
[000001][thread-0] starting
[000001][thread-0] obtained sum: 87192.184028
[000001][thread-0] finishing
[000001][thread-7] starting
[000001][thread-7] obtained sum: -281517.331236
[000001][thread-7] finishing
[000002][thread-4] starting
[000002][thread-4] obtained sum: -504592.070318
[000002][thread-4] finishing
[000002][thread-6] starting
[000002][thread-6] obtained sum: -388960.118190
[000002][thread-6] finishing
[000003][thread-2] starting
[000003][thread-2] obtained sum: -36177.722467
[000003][thread-2] finishing
```

En la captura de arriba, vamos a analizar si hace los productos escalares parciales de una forma correcta.

Cojamos el thread 1, al que se han asignado $v1[2]$ y $v2[2]$.

$v1[2] = 596.531599$

$v2[2] = 91.247875$



Scientífica

$$596,531599 \times 91,247875 =$$

54.432,240779102125

Por alguna razón, el resultado tan solo es correcto hasta el tercer decimal, pero vamos a darlo por bueno de momento.

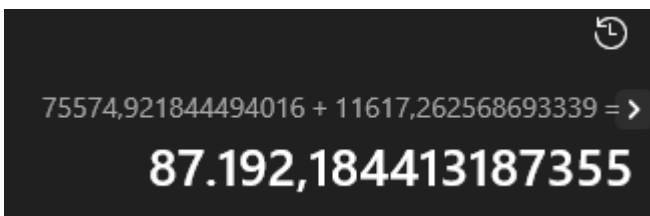
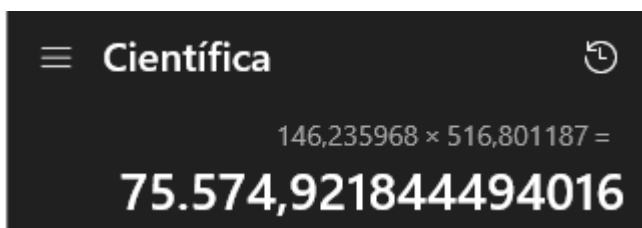
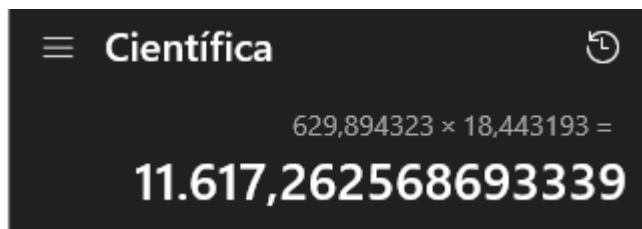
Por curiosidad, vamos a coger al thread 1, que hace dos pares de valores.

$v1[0] = 629.894323$

$v2[0] = 18.443193$

$v1[1] = 146.235968$

$v2[1] = 516.801187$



Mirando el resultado del shell, con varios pares también funcionaria el programa, obviando el error en los decimales. Hasta esa parte, el shell funciona correctamente.

```
[000003][main] thread 0 joined
[000003][main] waiting for thread 1
[000003][main] thread 1 joined
[000003][main] waiting for thread 2
[000003][main] thread 2 joined
[000003][main] waiting for thread 3
[000003][main] thread 3 joined
[000003][main] waiting for thread 4
[000003][main] thread 4 joined
[000003][main] waiting for thread 5
[000003][main] thread 5 joined
[000003][main] waiting for thread 6
[000003][main] thread 6 joined
[000003][main] waiting for thread 7
[000003][main] thread 7 joined
[000003][main] results:
Suma de los hilos (sin el hilo main): 0.000000
[000003][main] --> parallel sum: 0.000000
[000003][main] --> sequential sum: -205448.692439
[000003][main] --> not equal, WRONG!!
[estudiante_ic2@ic2-centos-vm trabajo3]$
```

Vemos que junta los resultados de cada hilo, pero el main tiene problemas para acceder a la última pareja de números (1 en este caso).

```
double sum_main = 0;

    for (int i = main_position; i < n_elems; i++){
        printf("valor: %f %d\n", v1[i], i);
        printf("valor: %f %d\n", v2[i], i);
        sum_main += v1[i]*v2[i];
    }
```

Vamos a imprimir los valores a ver si hay algún conflicto o es más adelante.

```
[000001][main] thread 7 joined
valor: 255.478494 9
valor: 134.135643 9
[000001][main] results:
Suma de los hilos (sin el hilo main): 134.135643
[000001][main] --> parallel sum: 134.135643
[000001][main] --> sequential sum: -1003634.006578
[000001][main] --> not equal, WRONG!!
[estudiante_ic2@ic2-centos-vm trabajo3]$
```

El main accede de forma correcta a los valores, ponemos más printf para intentar detectar el problema.

```
double sum_main = 0;


    for (int i = main_position; i < n_elems; i++){
        printf("valor: %f %d\n", v1[i], i);
        printf("valor: %f %d\n", v2[i], i);
        sum_main += v1[i]*v2[i];
    }

    printf("Resultado del thread main: %f\n", sum_main);

    printf("[%06ld][%s] results:\n", time_in_millis()-initial_time, __func__);
    //double parallel_sum=0;

    for(int i=0; i<n_threads; i++) {
        R += thread_ctxs[i].sum; // puede que el error este aqui
        printf("Valor de R: %f (iteracion %d)\n", R, i);
    }

    printf("Suma de los hilos (sin el hilo main): %f\n", R);
    R += sum_main;
```

```
Abrir   parallel_dot.c
~/grupoRAR/trabajo3

#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <sys/time.h>
#include <math.h>
#include <pthread.h>

#include "error.h"

#define MAX_RAND_VALUE 1000
#define ZERO_DIFF ((double) 1e-9)

unsigned long initial_time;

// variable global
double R = 0;
```

Antes habíamos puesto int R, hacemos ese cambio en adición.

```
[000002][main] waiting for thread 6
[000002][main] thread 6 joined
[000002][main] waiting for thread 7
[000002][main] thread 7 joined
valor: 333.674240 9
valor: -611.176712 9
Resultado del thread main: -203933.924733
[000002][main] results:
Valor de R: -279553.019670 (iteracion 0)
Valor de R: -551823.490316 (iteracion 1)
Valor de R: -177351.271200 (iteracion 2)
Valor de R: -819659.577874 (iteracion 3)
Valor de R: -1130313.295548 (iteracion 4)
Valor de R: -1051135.602691 (iteracion 5)
Valor de R: -1755365.854248 (iteracion 6)
Valor de R: -1139070.193520 (iteracion 7)
Resultado de los hilos (sin el hilo main): -1139070.193520
[000002][main] --> parallel sum: -1343004.118253
[000002][main] --> sequential sum: -1343004.118253
[000002][main] --> are equal, RIGHT!
[estudiante_ic2@ic2-centos-vm trabajo3]$
```

Ahora estaría bien en principio.

Añadimos ahora el bucle de 10000 iteraciones y observamos si hay errores. Para ello, vamos a quitar muchos printf, esencialmente dejando el printf que nos dice si R y R2 son diferentes.

```
int main(int argc, char *argv[])
{

    for (int i = 0; i<10000; i++){ // inicio del for de 10000 iteraciones

        printf("[%06ld][%s] --> parallel sum: %f\n",time_in_millis()-initial_time,__func__, R);

        //double sequential_sum=0;
        double R2 = 0;
        for(int i=0; i<n_elems; i++) R2 += v1[i]*v2[i];
        printf("[%06ld][%s] --> sequential sum: %f\n",time_in_millis()-initial_time,__func__, R2);
        printf(
            "[%06ld][%s] --> %s\n",
            time_in_millis()-initial_time,
            __func__,
            (fabs(R-R2)<ZERO_DIFF? "are equal, RIGHT!" : "not equal, WRONG!!")
        );

        return EXIT_SUCCESS;
    } // fin del for
```

Básicamente quitamos todos los printf menos esos.

Antes de eso, nos aseguramos de una cosa.

```
[estudiante_ic2@ic2-centos-vm trabajo3]$ ./parallel_dot 8
[000000][main] thread 0 joined
[000000][main] thread 1 joined
[000000][main] thread 2 joined
[000000][main] thread 3 joined
[000000][main] thread 4 joined
[000000][main] thread 5 joined
[000000][main] thread 6 joined
[000000][main] thread 7 joined
[000000][main] --> parallel sum: -1495142.636672
[000000][main] --> sequential sum: -1298611.447242
[000000][main] --> not equal, WRONG!!
Resultado: -1495142.636672
[estudiante_ic2@ic2-centos-vm trabajo3]$
```

Al poner longitud 8, da un resultado erróneo. Creemos que es porque el main hace la misma operación que el thread 7. Para ello, vamos a implementar un condicional.

```

double sum_main = 0;

if (n_elems == 8){
    sum_main = 0;
}

else{

for (int i = main_position; i < n_elems; i++){
    //printf("valor: %f %d\n", v1[i], i);
    //printf("valor: %f %d\n", v2[i], i);
    sum_main += v1[i]*v2[i];
}

}

//printf("Resultado del thread main: %f\n", sum_main);

```

Añadimos esta condición.

```

[estudiante_ic2@ic2-centos-vm trabajo3]$ ./parallel_dot 8
[000000][main] thread 0 joined
[000000][main] thread 1 joined
[000000][main] thread 2 joined
[000000][main] thread 3 joined
[000000][main] thread 4 joined
[000000][main] thread 5 joined
[000000][main] thread 6 joined
[000000][main] thread 7 joined
[000000][main] --> parallel sum: 508893.318387
[000000][main] --> sequential sum: 508893.318387
[000000][main] --> are equal, RIGHT!
Resultado: 508893.318387
[estudiante_ic2@ic2-centos-vm trabajo3]$

```

Volvemos con el for de 1000 iteraciones.

Antes no me hacía el for porque el return EXITSUCCESS estaba dentro, por eso solo se ejecutaba una vez.

```

        //printf("%d", sizeof("Resultado: %f"));
        // return EXIT_SUCCESS;

    }

    return EXIT_SUCCESS;
    // fin del for

}

```



```

        // return EXIT_SUCCESS;

    }
    // fin del for

    return EXIT_SUCCESS;
}
}

```

```

gcc -o parallel_dot parallel_dot.o error.o -lpthread -std=gnu99
[estudiante_ic2@ic2-centos-vm trabajo3]$ ./parallel_dot 8
[000001][main] thread 0 joined
[000001][main] thread 1 joined
[000001][main] thread 2 joined
[000001][main] thread 3 joined
[000001][main] thread 4 joined
[000001][main] thread 5 joined
[000001][main] thread 6 joined
[000001][main] thread 7 joined
[000001][main] --> parallel sum: -314918.334905
[000001][main] --> sequential sum: -314918.334905
[000001][main] --> are equal, RIGHT!
Resultado: -314918.334905
[000000][main] thread 0 joined
[000000][main] thread 1 joined
[000000][main] thread 2 joined
[000000][main] thread 3 joined
[000000][main] thread 4 joined
[000000][main] thread 5 joined
[000000][main] thread 6 joined
[000000][main] thread 7 joined
[000000][main] --> parallel sum: -629836.669810
[000000][main] --> sequential sum: -314918.334905
[000000][main] --> not equal, WRONG!!
Resultado: -629836.669810

```

Como no podíamos apreciar bien el output con 1000 iteraciones, lo cambiamos a 5. De esta forma vimos que la primera vez el programa da un resultado correcto, pero a partir de ahí no funciona.

```

void* thread(void* thread_arg) // se modifica para adaptarse al producto escalar
{
    thread_ctx* ctx=(thread_ctx*) thread_arg;

    //printf("[%06ld][%s-%u] starting\n",time_in_millis()-initial_time,__func__,ctx->id);
    //printf("[%06ld][%s-%u] adding %d elements starting at position %d:\n",time_in_millis()-initial_time,__func__,ctx->id,ctx->elems);

    //ctx->sum=0;
    // R = 0;
    for(int i=ctx->position; i<ctx->position+ctx->elems; i++)
    {
        //printf("[%06ld][%s-%u] --> adding element %d: %f\n",time_in_millis()-initial_time,__func__,ctx->id,i,ctx->vector[i]);
        R += ctx->vector1[i]*ctx->vector2[i];
    }

    //printf("[%06ld][%s-%u] obtained sum: %f\n",time_in_millis()-initial_time,__func__,ctx->id,ctx->sum);
    //printf("[%06ld][%s-%u] finishing\n",time_in_millis()-initial_time,__func__,ctx->id);
    return thread_arg;
}

```

```

    }
    //printf("Resultado del thread main: %f\n", sum_main);

    // printf("[%06ld][%s] results:\n",time_in_millis()-initial_time,__func__);
    //double parallel_sum=0;

    /*
    for(int i=0; i<n_threads; i++) {
        //R += thread_ctxs[i].sum; // puede que el error este aqui
        printf("Valor de R: %f (iteracion %d)\n", R, i);
    }
    */

    //printf("Resultado de los hilos (sin el hilo main): %f\n", R);
    R += sum_main;

```

Para comprobar con el mutex, nos fijamos en los apuntes que se utilizaban en funciones externas al main. Por eso cambiamos como estaba formado en un principio, que se añadía la suma al struc y luego en el main se sumaba a R, haciendo que ahora en la función se sumará directamente a R.

```

}

srand(time(NULL)+i);
double v1[n_elems]; // VECTOR 1
double v2[n_elems]; // VECTOR 2

/*printf(
    "[%06ld][%s] random vector (%d elements):\n",
    time_in_millis()-initial_time,
    __func__,
    n_elems
); */

// CREAMOS LOS VECTORES CON VALORES ALEATORIOS
//printf("Valores del VECTOR 1\n");
for(int i=0; i<n_elems; i++)
{
    v1[i]=((double)(2*MAX RAND_VALUE)*rand())/((double) RAND_MAX)-MAX RAND_VALUE;
    /*printf(
        "[%06ld][%s] --> element %d: %f\n",
        time_in_millis()-initial_time,
        __func__,i,v1[i]
    ); */
}

```

Otra cosa de la que nos fijamos es que con la semilla que había daba todo el rato los mismos valores en cada iteración, por eso para que sea más aleatorios aprovechamos el contador del for *i* en la semilla para que se le sume en cada iteración.

```

[estudiante_ic2@ic2-centos-vm trabajo3]$ ./parallel_dot 8
[000000][main] thread 0 joined
[000000][main] thread 1 joined
[000000][main] thread 2 joined
[000000][main] thread 3 joined
[000000][main] thread 4 joined
[000000][main] thread 5 joined
[000000][main] thread 6 joined
[000000][main] thread 7 joined
[000000][main] --> parallel sum: -661856.491384
[000000][main] --> sequential sum: -661856.491384
[000000][main] --> are equal, RIGHT!
Resultado: -661856.491384
[000000][main] thread 0 joined
[000000][main] thread 1 joined
[000000][main] thread 2 joined
[000000][main] thread 3 joined
[000000][main] thread 4 joined
[000000][main] thread 5 joined
[000000][main] thread 6 joined
[000000][main] thread 7 joined
[000000][main] --> parallel sum: -1059685.864091
[000000][main] --> sequential sum: -1059685.864091
[000000][main] --> are equal, RIGHT!
Resultado: -1059685.864091

```

```

for (int i = 0; i < 100000; i++){ // inicio del for de 10000 iteraciones

```

```

    R = 0;

```

```

    initial_time=time_in_millis();

```

```

    char* program_name=strrchr(argv[0], '/')+1;

```

```

    const unsigned int n_threads=8;

```

```

    int n_elems=((argc>1)? atoi(argv[1]): 0);

```

```

    if(n_elems<n_threads)

```

```

    {

```

```

        print_error_warning_msg(

```

```

            "ERROR",

```

```

            __func__,

```

```

            __FILE__,

```

```

            __LINE__,

```

```

            "how many elements do you want to sum? At least must be %d!",

```

```

            n_threads

```

```

        );

```

```

// variables globales
double R = 0;
pthread_mutex_t cerrojo;
int horrible = 0;

if (R != R2) {
    horrible++;
}

// SEGMENTO DE ARCHIVOS

int fichero;
int contarbyte = 0;
int lenr = sizeof(R) + sizeof("Resultado: %f") + contarbyte;
char preresultado[50];
sprintf(preresultado, "Resultado: %f", R);
int counter = 0;
while (preresultado[counter] != '\0'){
    counter++;
}
char resultado[counter];
fichero = open("resultado.txt", O_RDWR | O_CREAT | O_TRUNC, 0644);
sprintf(resultado, "Resultado: %f", R);
write(fichero, resultado, sizeof(resultado));
close(fichero);
fichero = open("resultado.txt", O_RDONLY);
read(fichero, resultado, sizeof(resultado));
close(fichero);
printf("%s\n", resultado);
//int sizeprueba = sizeof(resultado);
//printf("%d", sizeof(R));
//printf("%d", sizeof("Resultado: %f"));
// return EXIT_SUCCESS;

}
// fin del for

printf("Errores: %d\n", horrible);

```

Se creó una variable horrible que se incrementa tras cada error.

```

[000001][main] --> sequential sum: -241333.506838
[000001][main] -->
Resultado: -241333.506838
Errores: 34877
[estudiante_ic2@ic2-centos-vm trabajo3]$

```

Ese fue el resultado con 100000 iteraciones. Vamos a probar con 1000.

```
[000000][main] --> sequential sum: 1117600.721504
[000000][main] -->
Resultado: 1117600.721504
Errores: 338
[estudiante_ic2@ic2-centos-vm trabajo3]$
```

Como se puede ver hay errores en el código ya que se incrementa el valor y salía Wrong en algunos casos.

```
unsigned long initial_time;

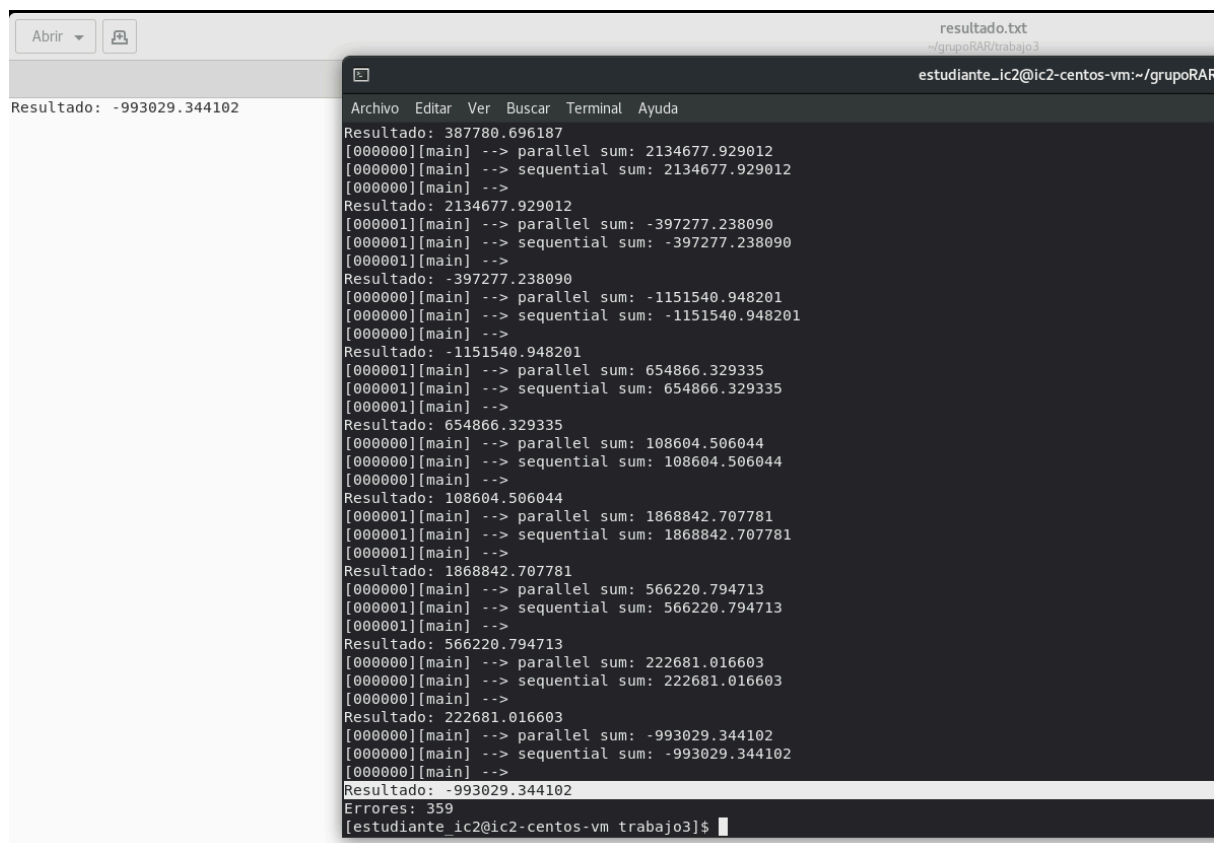
// variables globales
double R = 0;
pthread_mutex_t cerrojo;
//int horrible = 0;

// ...
for(int i=ctx->position; i<ctx->position+ctx->elems; i++)
{
    //printf("[%06ld][%s-%u] --> adding element %d: %f\n",time_
    pthread_mutex_lock(&cerrojo);
    R += ctx->vector1[i]*ctx->vector2[i];
    pthread_mutex_unlock(&cerrojo);
    //printf("%f\n", R);
}

printf(
    "[%06ld][%s] --> %s\n",
    time_in_millis()-initial_time,
    __func__,
    (fabs(R-R2)<ZERO_DIFF? "" : "not equal, WRONG!!")
);

if (R-R2 != 0) {
    horrible++;
    // exit(0);
}
```

```
[000000][main] --> sequential sum: 1117600.721504
[000000][main] -->
Resultado: 222681.016603
[000000][main] --> parallel sum: -993029.344102
[000000][main] --> sequential sum: -993029.344102
[000000][main] -->
Resultado: -993029.344102
Errores: 359
[estudiante_ic2@ic2-centos-vm trabajo3]$
```

```
Archivo Editar Ver Buscar Terminal Ayuda
Resultado: 387780.696187
[000000][main] --> parallel sum: 2134677.929012
[000000][main] --> sequential sum: 2134677.929012
[000000][main] -->
Resultado: 2134677.929012
[000001][main] --> parallel sum: -397277.238090
[000001][main] --> sequential sum: -397277.238090
[000001][main] -->
Resultado: -397277.238090
[000000][main] --> parallel sum: -1151540.948201
[000000][main] --> sequential sum: -1151540.948201
[000000][main] -->
Resultado: -1151540.948201
[000001][main] --> parallel sum: 654866.329335
[000001][main] --> sequential sum: 654866.329335
[000001][main] -->
Resultado: 654866.329335
[000000][main] --> parallel sum: 108604.506044
[000000][main] --> sequential sum: 108604.506044
[000000][main] -->
Resultado: 108604.506044
[000001][main] --> parallel sum: 1868842.707781
[000001][main] --> sequential sum: 1868842.707781
[000001][main] -->
Resultado: 1868842.707781
[000000][main] --> parallel sum: 566220.794713
[000001][main] --> sequential sum: 566220.794713
[000001][main] -->
Resultado: 566220.794713
[000000][main] --> parallel sum: 222681.016603
[000000][main] --> sequential sum: 222681.016603
[000000][main] -->
Resultado: 222681.016603
[000000][main] --> parallel sum: -993029.344102
[000000][main] --> sequential sum: -993029.344102
[000000][main] -->
Resultado: -993029.344102
Errores: 359
[estudiante_ic2@ic2-centos-vm trabajo3]$
```

Se implementa el mutex para que no ocurra ninguna excepción crítica. Pero igualmente nos fijamos que la variable horrible nos avisa de que ocurrieron muchos errores. Pero si se mira todo lo que devuelve el output no hay ningún error.

Esto se puede comprobar si se descomenta el `exit(0)`; ya que esto para el proceso cuando ocurra un error, lo que nos hace ver que se para en un caso que es igual.

```
//double sequential_sum=0;
double R2 = 0;
for(int i=0; i<n elems; i++) R2 += v1[i]*v2[i];
printf("[%06ld][%s] --> sequential sum: %f\n",time_in_millis()-initial_time,__func__, R2);
printf(
    "[%06ld][%s] --> %s\n",
    time_in_millis()-initial_time,
    __func__,
    (fabs(R-R2)<ZERO_DIFF? "" : "not equal, WRONG!!")
);

if (fabs(R-R2) >= ZERO_DIFF) {
    horrible++;
    exit(0);
}
```

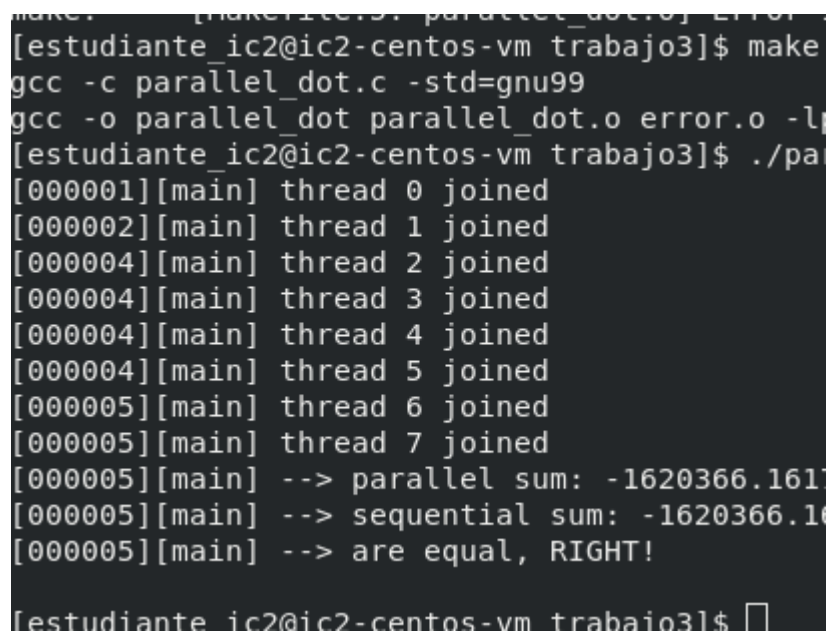
```
Resultado: -1191997.185685
[000000][main] --> parallel sum: 152843.807330
[000000][main] --> sequential sum: 152843.807330
[000000][main] -->
Resultado: 152843.807330
[000001][main] --> parallel sum: -537322.471411
[000001][main] --> sequential sum: -537322.471411
[000001][main] -->
Resultado: -537322.471411
Errores: 0
[estudiante_ic2@ic2-centos-vm trabajo3]$
```

Nos fijamos que en el print para realizar la operación se hace una operación distinta a la que usamos en el if anteriormente. Por lo que, para intentar solucionar el problema existente reutilizamos esa operación en el if pero indicando el caso contrario a ese para que se sume cuando ocurra un Wrong. Podemos observar que ahora no hubo ningún error y no se incrementó la variable *horrible*.

Sección de archivos:

Crear archivo que guarde el resultado:

```
int fichero;
fichero = creat("resultado.txt", O_RDWR);
char *resultado[50];
write(fichero, resultado, ("Resultado:%f\n", R));
close(fichero);
fichero = open("resultado.txt", O_RDWR);
read(fichero, resultado, sizeof(resultado));
close(fichero);
printf("%s\n", resultado);
```



```
make: [makefile:3: parallel_dot.o] Error
[estudiante_ic2@ic2-centos-vm trabajo3]$ make
gcc -c parallel_dot.c -std=gnu99
gcc -o parallel_dot parallel_dot.o error.o -lpthread
[estudiante_ic2@ic2-centos-vm trabajo3]$ ./parallel_dot
[000001][main] thread 0 joined
[000002][main] thread 1 joined
[000004][main] thread 2 joined
[000004][main] thread 3 joined
[000004][main] thread 4 joined
[000004][main] thread 5 joined
[000005][main] thread 6 joined
[000005][main] thread 7 joined
[000005][main] --> parallel sum: -1620366.1617
[000005][main] --> sequential sum: -1620366.1617
[000005][main] --> are equal, RIGHT!
[estudiante_ic2@ic2-centos-vm trabajo3]$
```

Se me crea el archivo, pero no puedo comprobar si realmente se escribe porque no me lee el archivo.

```
int fichero;  
char resultado[50];  
fichero = creat("resultado.txt", 0_RDWR);  
sprintf(resultado, "%f", R);  
write(fichero, resultado, sizeof(resultado));  
close(fichero);  
fichero = open("resultado.txt", 0_RDONLY);  
read(fichero, resultado, sizeof(resultado));  
close(fichero);  
printf("Resultado:%s\n", resultado);  
  
return EXIT_SUCCESS;
```

Me faltaba tener el resultado pasado de double a char

```
[estudiante_ic2@ic2-centos-vm trabajos]$ ./parallel_
[000001][main] thread 0 joined
[000001][main] thread 1 joined
[000002][main] thread 2 joined
[000002][main] thread 3 joined
[000002][main] thread 4 joined
[000002][main] thread 5 joined
[000002][main] thread 6 joined
[000002][main] thread 7 joined
[000002][main] --> parallel sum: -1096385.901751
[000002][main] --> sequential sum: -1096385.901751
[000002][main] --> are equal, RIGHT!
Resultado: -1096385.901751
```

Lo escribe, pero hay un pequeño problema:

```
-1096385.901751\00\D0"0006\FB\FF00YF\00\00\98\BA*8\FF~\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00
```

Esto es lo que hay dentro del documento, que no se que es.

He buscado información de internet, concretamente de la página de IBM:

<https://www.ibm.com/docs/fr/zos/2.4.0?topic=functions-creat-create-new-file-rewrite-existin-g-one>

Cambié el creat por su equivalente de open pero añadiendo el int mode 0644, ya que no lo encontré en los pdf más allá de que servía para dar permisos, por lo que decidí buscarlo en internet y vi que si ponías ese int mode 0644 si que me daba permisos de lectura y escritura del documento, cosa que antes no pasaba

<https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/>

Luego lo busqué de manera más concreta el 0644 y me usé esta página:

<https://stackoverflow.com/questions/18415904/what-does-mode-t-0644-mean>

```
int fichero;
int lenr = sizeof(R) + sizeof("Resultado: %f");
char resultado[22];
fichero = open("resultado.txt", O_RDWR | O_CREAT | O_TRUNC, 0644);
sprintf(resultado, "Resultado: %f", R);
write(fichero, resultado, sizeof(resultado));
close(fichero);
fichero = open("resultado.txt", O_RDONLY);
read(fichero, resultado, sizeof(resultado));
close(fichero);
printf("%s\n", resultado);
int sizeprueba = sizeof(resultado);
printf("%d", sizeof(R));
printf("%d", sizeof("Resultado: %f"));
return EXIT_SUCCESS;
```

Cambié un poco la longitud de resultado y esto sale en el fichero:

```
Resultado: 1124300.167
```

Le faltan 3 decimales, y después de probar con resultados positivos y negativos caímos en la cuenta de que si es negativo tiene un byte más y que el punto también es un byte más, por lo que ideamos con un `while` y un `preresultado` para comprobar cuantos bytes había antes del carácter nulo:

```

int fichero;
int contarbyte = 0;
int lenr = sizeof(R) + sizeof("Resultado: %f") + contarbyte;
char preresultado[50];
sprintf(preresultado, "Resultado: %f", R);
int counter = 0;
while (preresultado[counter] != '\\0'){
    counter++;
}
char resultado[counter];
fichero = open("resultado.txt", O_RDWR | O_CREAT | O_TRUNC, 0644);
sprintf(resultado, "Resultado: %f", R);
write(fichero, resultado, sizeof(resultado));
close(fichero);
fichero = open("resultado.txt", O_RDONLY);
read(fichero, resultado, sizeof(resultado));
close(fichero);
printf("%s\\n", resultado);
//int sizeprueba = sizeof(resultado);
/printf("%d", sizeof(R));
/printf("%d", sizeof("Resultado: %f"));
return EXIT SUCCESS;

```

Con número negativo:

```

[estudiante_ic2@ic2-centos-vm trabajo3]$ ./parallel_dot 10
[000003][main] thread 0 joined
[000003][main] thread 1 joined
[000003][main] thread 2 joined
[000003][main] thread 3 joined
[000004][main] thread 4 joined
[000004][main] thread 5 joined
[000004][main] thread 6 joined
[000004][main] thread 7 joined
[000004][main] --> parallel sum: -528915.822844
[000004][main] --> sequential sum: -528915.822844
[000004][main] --> are equal, RIGHT!
Resultado: -528915.822844
[estudiante_ic2@ic2-centos-vm trabajo3]$

```

Abrir ▾



resultado.txt

~/trabajo3

Resultado: -528915.822844

Y ahora uno positivo:

```
[estudiante_ic2@ic2-centos-vm trabajo3]$ ./parallel_dot 10
[000004][main] thread 0 joined
[000004][main] thread 1 joined
[000004][main] thread 2 joined
[000004][main] thread 3 joined
[000005][main] thread 4 joined
[000005][main] thread 5 joined
[000005][main] thread 6 joined
[000005][main] thread 7 joined
[000005][main] --> parallel sum: 424092.463959
[000005][main] --> sequential sum: 424092.463959
[000005][main] --> are equal, RIGHT!
Resultado: 424092.463959
[estudiante_ic2@ic2-centos-vm trabajo3]$
```

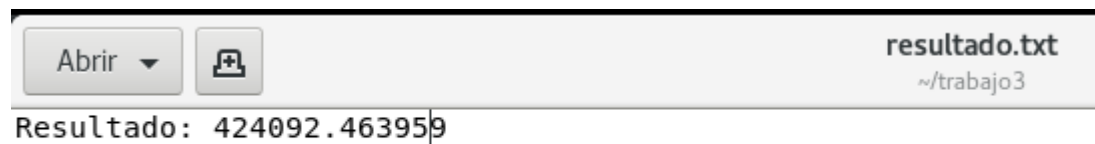
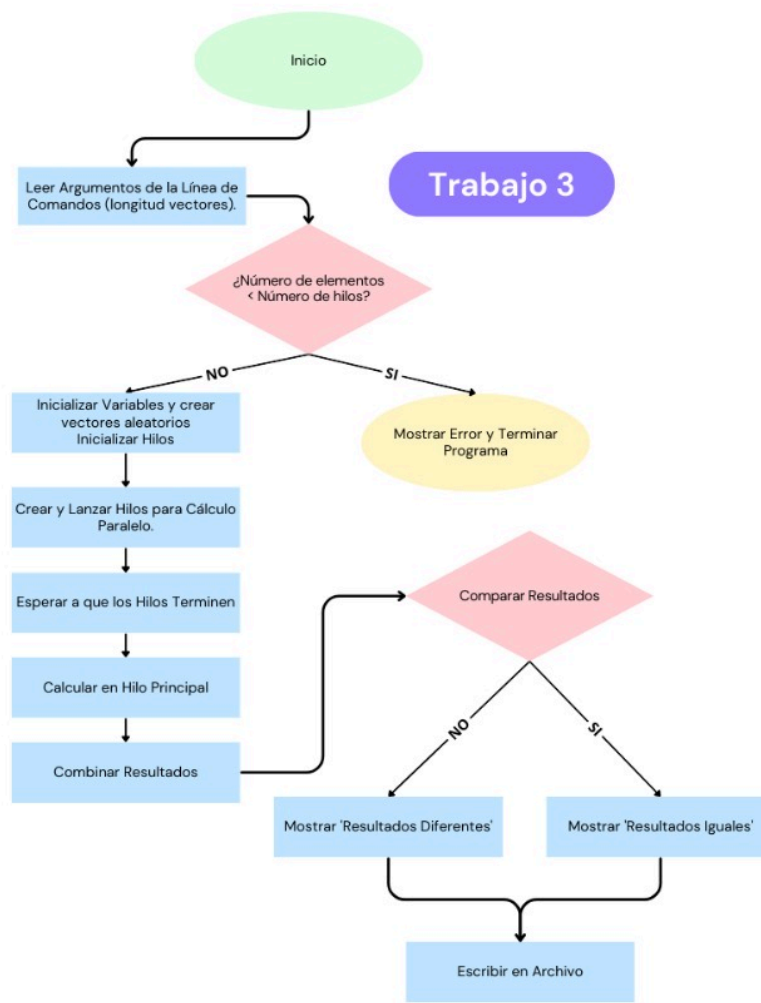


DIAGRAMA DE FLUJO PLANTEADO:



PLANTEAMIENTO DEL TRABAJO:

- **Nesta:** Implementación del apartado de archivos del trabajo (6.2), ayuda general en el trabajo, diagrama de flujo y resolver algunos errores.
- **Enrique:** Realizó gran parte del ejercicio de entrenamiento, ayudó tanto en 6.1 a Ayoze para resolver errores que permanecían en el código, como en 6.2 a Nesta, a la hora de calcular el tamaño que se debe de imprimir en el archivo(que no imprimiera \0).
- **Ayoze:** Hizo la mayoría del apartado 6.1, junto con la ayuda de Enrique. Con esto se refiere a la modificación del código para hacerlo acorde al producto escalar, así como la perfilación del output.