

## Hoja de Trabajo #1

**Fecha de Entrega:** 11 de Agosto, 2023.

**Descripción:** En esta hoja de trabajo empezará a familiarizarse con las directivas, cláusulas y funciones de OpenMP haciendo uso de ellas en pequeños ejercicios que demostrarán su funcionamiento. Explorará el uso de OpenMP con 2 pequeños ejercicios introductorios para familiarizarse con el API y luego creará un programa que resuelve y ejecuta un método numérico y lo transformará de secuencial a paralelo.

**Entregables:** Deberá entregar un documento con las respuestas a las preguntas planteadas en cada ejercicio (incluyendo diagramas o screenshots si es necesario), junto con todos los archivos de código que programe debidamente comentados e identificados.

**Materiales:** necesitará una máquina virtual con Linux.

### Contenido:

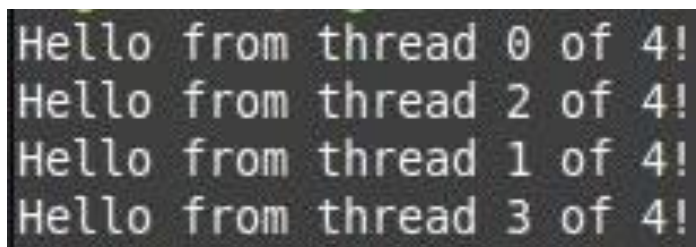
#### Ejercicio 1 (10 puntos)

Verifique y asegúrese que el compilador GCC está disponible en su sistema. Luego, escriba y compile un programa en C llamado "hello\_omp.c" que, haciendo uso del api de OpenMP, imprima N veces el mensaje "Hello from thread <número de thread> of <cantidad de threads> !".

El mensaje debe incluir el *número de thread* del thread realizando el *printf* y la *cantidad de threads* que su programa ejecutará. La cantidad de threads debe ser ingresada desde línea de comando al ejecutar el programa. También, valide que si el número de threads no fue ingresado desde línea de comando, su programa automáticamente utilizará 10 threads como valor *default*.

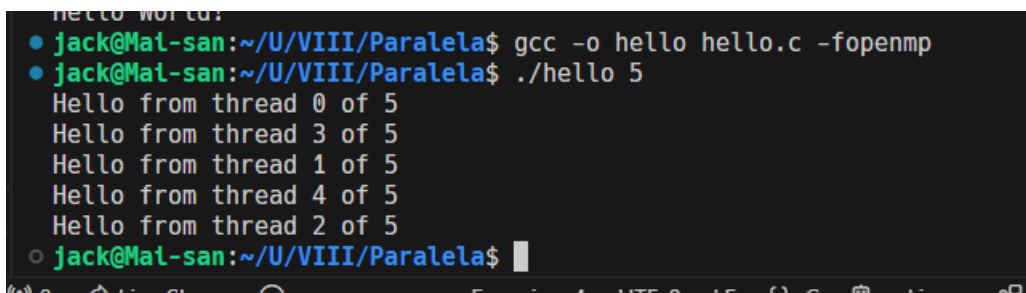
**NO** utilice ciclos *for* en su implementación.

Ejemplo del resultado esperado:



```
Hello from thread 0 of 4!  
Hello from thread 2 of 4!  
Hello from thread 1 of 4!  
Hello from thread 3 of 4!
```

Recuerde agregar screenshots de la ejecución de sus programas.



```
hello world:  
• jack@Mai-san:~/U/VIII/Paralela$ gcc -o hello hello.c -fopenmp  
• jack@Mai-san:~/U/VIII/Paralela$ ./hello 5  
Hello from thread 0 of 5  
Hello from thread 3 of 5  
Hello from thread 1 of 5  
Hello from thread 4 of 5  
Hello from thread 2 of 5  
• jack@Mai-san:~/U/VIII/Paralela$
```

**PREGUNTA: ¿Por qué al ejecutar su código los mensajes no están desplegados en orden?**

La razón por la que los mensajes no están ordenados cuando se ejecuta el programa es por la forma de ejecución de OpenMP. Cuando se usa la directiva de ``#pragma omp parallel`` se crean varios threads que se ejecutan en paralelo y no se garantiza el orden que se ejecutan.

Cada thread llama a la función entonces para imprimirse de forma independiente y, dependiendo de muchos factores los threads que se ejecutan llevan cierto orden.

## **Ejercicio 2 (10 puntos)**

Copie el código del ejercicio # 1 y luego modifíquelo para que haga lo siguiente. Cree un programa en C llamado "hbd\_omp.c" que imprima el ID de cada thread y la cantidad de threads. Imprima los siguientes mensajes dependiendo de si el ID del thread es par o impar:

- ID Impar: "Feliz cumpleaños número <cantidad de threads>!".
- ID Par: "Saludos del hilo <id del thread>"

Al momento de ejecutar su programa envíe como parámetro de cantidad de threads su edad. Compílelo y ejecútelo con el comando:

```
./<nombre_ejecutable> <SU_EDAD>
```

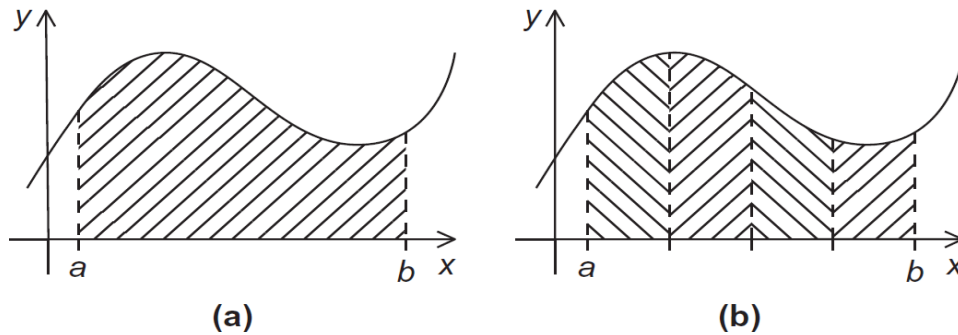
Ejemplo del resultado esperado:

```
Feliz cumpleaños número 10!  
Saludos del hilo 0  
Feliz cumpleaños número 10!  
Saludos del hilo 2  
Saludos del hilo 4  
Feliz cumpleaños número 10!  
Saludos del hilo 6  
Feliz cumpleaños número 10!  
Saludos del hilo 8  
Feliz cumpleaños número 10!
```

```
● jack@Mal-san:~/U/VIII/Paralela$ ./ejercicio2 22  
Saludos del hilo 2  
Saludos del hilo 8  
Feliz cumpleaños número 22!  
Feliz cumpleaños número 22!  
Saludos del hilo 4  
Saludos del hilo 10  
Saludos del hilo 6  
Feliz cumpleaños número 22!  
Feliz cumpleaños número 22!  
Feliz cumpleaños número 22!  
Saludos del hilo 18  
Feliz cumpleaños número 22!  
Feliz cumpleaños número 22!  
Saludos del hilo 0  
Feliz cumpleaños número 22!  
Saludos del hilo 14  
Saludos del hilo 16  
Saludos del hilo 20  
Feliz cumpleaños número 22!  
Feliz cumpleaños número 22!  
Feliz cumpleaños número 22!  
Saludos del hilo 12  
○ jack@Mal-san:~/U/VIII/Paralela$
```

**Ejercicio 3 (40 puntos)**

La “Regla Trapezoidal” o “Sumas de Riemann” es un método numérico que nos permite realizar aproximaciones sobre integrales que no tienen una solución directa con los métodos tradicionales de integrales (sustitución, por partes, etc...). Lo que el método nos dice es que podemos estimar el área bajo una curva dividiéndola en trapezoides de tamaño finito.



Para resolver este problema se define un intervalo  $(a, b)$  de los puntos sobre los cuales queremos encontrar la estimación del área bajo la curva. Luego lo dividiremos en  $n$  subintervalos iguales que conformaran nuestros trapezoides. Mientras mayor sea  $n$  mejor será la estimación. Entonces, para estimar el área bajo la curva de una función  $ff(x)$  debemos sumar el área del trapezoide  $T_0$  hasta  $T_n$ .

Dicha suma se puede expresar con la siguiente fórmula

$$\int_a^b ff(x) \approx \frac{h}{2} * ff(x_0) + ff(x_1) + ff(x_2) + ff(x_3) + \dots + ff(x_{n-1}) + \frac{h}{2} ff(x_n)$$

Donde  $h$  representa la base de cada trapezoide y se ve de la siguiente manera:

$$h = \frac{b - a}{n}$$

Y donde los inputs de nuestra función  $ff(x)$  se ven de la siguiente manera:

$$\begin{aligned} x_0 &= a \\ x_1 &= a + h \\ x_2 &= a + h + h = a + 2h \\ x_{n-1} &= a + (n - 1)h \\ x_n &= b \end{aligned}$$

Cree un archivo llamado “riemann.c” en donde codificará su programa. Investigue acerca de las “Sumas de Riemann” y sobre la “Regla trapezoidal” de ser necesario para mejorar su entendimiento sobre el programa que codificará. Su programa deberá contener la función “trapezoides” para calcular de forma secuencial una suma de riemann para una función  $ff(x)$  previamente definida. Los intervalos  $(a, b)$  para el cálculo deben ser ingresados desde línea de comando.

```
./<nombre_ejecutable> a b
```

Ejecute su programa, utilizando  $n = 10e6$  para las siguientes funciones y con los siguientes intervalos:

- $x^2$  (2, 10)
- $2x^3$  (3, 7)
- $\sin(x)$  (0, 1)

Imprima el resultado de su programa de la siguiente manera:

```
Con n = 10000000 trapezoides, nuestra aproximacion
de la integral de 3.000000 a 7.000000 es = 1159.999999998
```

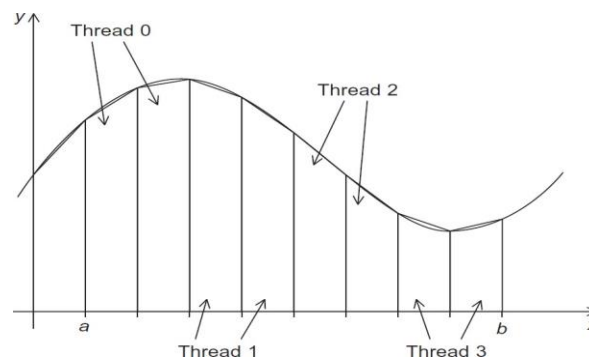
Recuerde agregar screenshots de la ejecución de sus programas.

```
• jack@Mat-san:~/U/VIII/Paralela$ gcc -o riemann riemann.c -lm
• jack@Mat-san:~/U/VIII/Paralela$ ./riemann 2 10
Con n = 10000000 trapezoides, nuestra aproximación de la integral de 2.00 a 10.00 para x^2 es = 330.666666667118
Con n = 10000000 trapezoides, nuestra aproximación de la integral de 2.00 a 10.00 para 2x^3 es = 4992.0000000005039
Con n = 10000000 trapezoides, nuestra aproximación de la integral de 2.00 a 10.00 para sin(x) es = 0.4229246925294
• jack@Mat-san:~/U/VIII/Paralela$ ./riemann 3 7
Con n = 10000000 trapezoides, nuestra aproximación de la integral de 3.00 a 7.00 para x^2 es = 105.333333333385
Con n = 10000000 trapezoides, nuestra aproximación de la integral de 3.00 a 7.00 para 2x^3 es = 1159.999999998317
Con n = 10000000 trapezoides, nuestra aproximación de la integral de 3.00 a 7.00 para sin(x) es = -1.7438947509438
• jack@Mat-san:~/U/VIII/Paralela$ ./riemann 0 1
Con n = 10000000 trapezoides, nuestra aproximación de la integral de 0.00 a 1.00 para x^2 es = 0.333333333334
Con n = 10000000 trapezoides, nuestra aproximación de la integral de 0.00 a 1.00 para 2x^3 es = 0.500000000000
Con n = 10000000 trapezoides, nuestra aproximación de la integral de 0.00 a 1.00 para sin(x) es = 0.4596976941318
○ jack@Mat-san:~/U/VIII/Paralela$
```

#### Ejercicio 4 (20 puntos)

Ahora que su programa “riemann.c” calcula serialmente una aproximación de la integral de una función  $f(x)$  necesitamos paralelizar este proceso. Para ello cree una copia de su programa y renómbrelo como “remann\_omp2.c”.

Para hacer esta conversión asumimos que manejamos muchos más trapezoides que threads. Vamos a dividir el total de trapezoides (data) dentro del número de threads, es decir, **división del dominio**. Asegúrese que el número de trapezoides sea múltiplo del número de threads.



Necesitará hacer uso de lo que llamaremos “parámetros locales” y para asegurarnos de que cada thread realice el trabajo correcto, debemos crear explícitamente los parámetros locales de cada uno:

- número local de trapezoides =  $n\_local = \text{trapezoides} / \text{threads}$
- valor inicial local del rango =  $a\_local = a + (ID\_thread * n\_local * ancho\_h)$
- valor final local del rango =  $b\_local = a\_local + (n\_local * ancho\_h)$

Ejemplo:

- $a = 0, b = 100, n = 500, \text{threads} = 4$
- $h = (100 - 0) / 500 = 0.2$
- $n\_local = 500 / 4 = 125$

ID	span = n_local*h offset = ID*span	a_local = a + offset	b_local = a_local + span
0	$0 * (125 * 0.2) = 0$	$0 + 0 = 0$	$0 + (125 * 0.2) = 25$
1	$1 * (125 * 0.2) = 25$	$0 + 25 = 25$	$25 + (125 * 0.2) = 50$
2	$2 * (125 * 0.2) = 50$	$0 + 50 = 50$	$50 + (125 * 0.2) = 75$
3	$3 * (125 * 0.2) = 75$	$0 + 75 = 75$	$75 + (125 * 0.2) = 100$

Realice los ajustes a su código para recibir como parámetro adicional la cantidad de threads a utilizar.

```
./<nombre_ejecutable> a b <cantidad de threads>
```

Además, cuando un thread ejecute la función “trapezoides” este debe calcular una parte de la integral e irlo añadiendo a una variable global (utilice la directiva **#pragma omp critical** en este punto). Despliegue el resultado en pantalla.

**NO** paralelice los ciclos *for* que pudiese tener adentro de su función.

```

jack@Mat-san:~/U/VIII/Paralela$ gcc -o riemann2 riemann_omp2.c -lm -fopenmp
jack@Mat-san:~/U/VIII/Paralela$ ./riemann2 2 10
Uso: ./riemann2 a b <cantidad de threads>
jack@Mat-san:~/U/VIII/Paralela$ ./riemann2 2 10 10
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 2.00 a 10.00 para x^2 es = 330.66666666666682
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 2.00 a 10.00 para 2x^3 es = 4991.99999999999945
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 2.00 a 10.00 para sin(x) es = 0.4229246925293
jack@Mat-san:~/U/VIII/Paralela$ ./riemann2 3 7 10
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 3.00 a 7.00 para x^2 es = 105.33333333333329
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 3.00 a 7.00 para 2x^3 es = 1160.00000000000036
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 3.00 a 7.00 para sin(x) es = -1.7438947509438
jack@Mat-san:~/U/VIII/Paralela$ ./riemann2 0 1 10
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 0.00 a 1.00 para x^2 es = 0.33333333333333
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 0.00 a 1.00 para 2x^3 es = 0.50000000000000
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 0.00 a 1.00 para sin(x) es = 0.4596976941319
jack@Mat-san:~/U/VIII/Paralela$

```

**PREGUNTA: ¿Por qué es necesario el uso de la directiva **#pragma omp critical**?**

Usando el `#pragma omp critical` nos aseguramos de que solo un thread a la vez pueda acceder y actualizar una variable, en este caso `suma_global`. Esto nos ayuda a prevenir race conditions y nos asegura que la suma se pueda realizar correctamente.



Ahora que ya tiene un programa que paraleliza la resolución de calcular una integral a través de la “Regla Trapezoidal” / “Sumas de Riemman” lo que haremos es evitar el uso de la directiva **#pragma omp critical**. Para ello cree una nueva versión del código que programó en el ejercicio 4 y renómbrelo a “riemann\_omp\_nocrit.c”.

En esta nueva iteración del programa, deberá agregar un arreglo global en donde almacene el calculo que realizó localmente cada uno de los threads. Finalmente, sume los resultados del arreglo para obtener el resultado final y despléguelo en pantalla.

```

● jack@Mat-san:~/U/VIII/Paralela$ gcc -o riemann3 riemann_omp_nocrit.c -lm -fopenmp
● jack@Mat-san:~/U/VIII/Paralela$ ./riemann3 2 10 10
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 2.00 a 10.00 para x^2 es = 330.6666666666682
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 2.00 a 10.00 para 2x^3 es = 4991.9999999999955
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 2.00 a 10.00 para sin(x) es = 0.4229246925293
● jack@Mat-san:~/U/VIII/Paralela$ ./riemann3 3 7 10
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 3.00 a 7.00 para x^2 es = 105.3333333333329
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 3.00 a 7.00 para 2x^3 es = 1160.0000000000036
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 3.00 a 7.00 para sin(x) es = -1.7438947509438
● jack@Mat-san:~/U/VIII/Paralela$ ./riemann3 0 1 10
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 0.00 a 1.00 para x^2 es = 0.3333333333333
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 0.00 a 1.00 para 2x^3 es = 0.5000000000000
Con n = 10000000 trapezoides y 10 threads, nuestra aproximación de la integral de 0.00 a 1.00 para sin(x) es = 0.4596976941319
○ jack@Mat-san:~/U/VIII/Paralela$

```

**PREGUNTA: ¿Qué diferencia hay entre usar una variable global para añadir los resultados a un arreglo?**

	Variable Global	Arreglo local
<b>COONCURRENCIA</b>	Si varios hilos intentan actualizar la misma variable global simultáneamente, puede producirse race conditions	Para evitar las races conditions, necesitarías utilizar mecanismos de sincronización, como <b>#pragma omp critical</b> , que permiten que solo un hilo acceda a la variable a la vez.
<b>SINCRONIA</b>	Al asignar una posición del arreglo para cada hilo, puedes evitar las condiciones de carrera, ya que cada hilo solo actualiza su posición en el arreglo y no afecta a los demás hilos	: No se necesita sincronización entre los hilos durante la actualización del arreglo, ya que cada hilo tiene su propia posición en el arreglo. Esto puede llevar a una mayor eficiencia y escalabilidad.
<b>POST PROCESAMIENTO</b>	No necesita	Necesitarás un paso adicional para sumar los valores del arreglo

Universidad del Valle de Guatemala  
Computación Paralela y Distribuida  
Docente: Sebastián Galindo  
Semestre 2, 2023

