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

**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**:

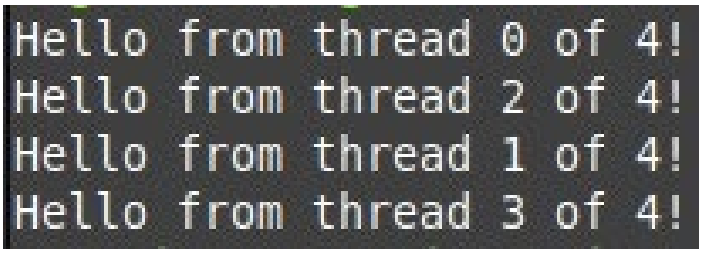
**Ejercicio 1 (10 puntos)**

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:




Recuerde agregar screenshots de la ejecución de sus programas.

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

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

**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:

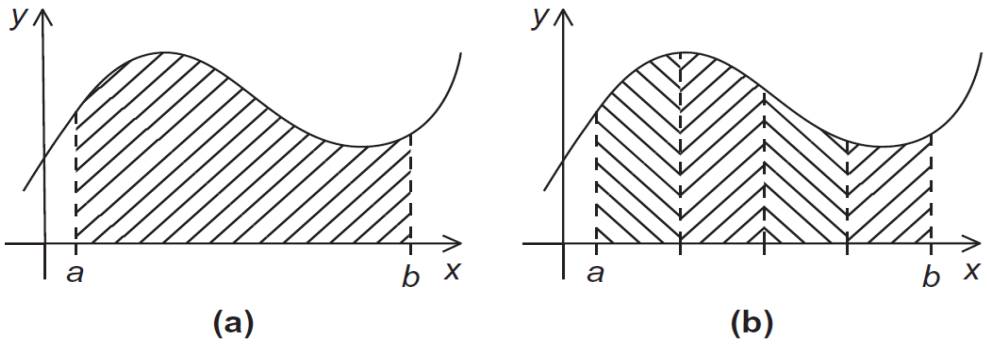
```
./<nombre_ejecutable> <SU_EDAD>
```

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!
```

**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 $f(x)$ debemos sumar el área del trapezoide $T_0$ hasta $T_n$.

Dicha suma se puede expresar con la siguiente fórmula

$$\int_a^b f(x) \approx \frac{h}{2} * f(X_0) + f(X_1) + f(X_2) + f(X_3) + \cdots + f(X_{n-1}) + \frac{h}{2} f(X_n)$$

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 $f(x)$ se ven de la siguiente manera:

$$X_0 = a$$
$$X_1 = a + h$$
$$X_2 = a + h + h = a + 2h$$
$$X_{n-1} = a + (n - 1)h$$
$$X_n = b$$

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 $f(x)$ 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 = 1159.9999999998
```

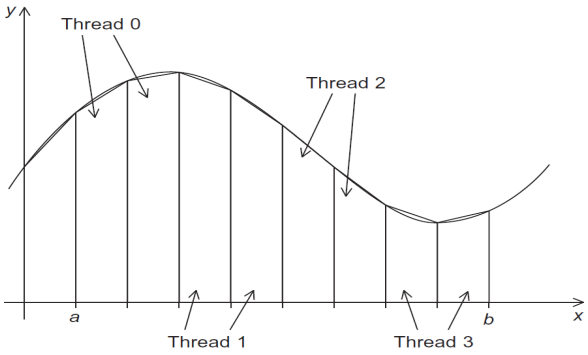
Recuerde agregar screenshots de la ejecución de sus programas.

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

**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 = trapezoides / 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, threads = 4
- h = (100-0)/500 = 0.2
- n_local = 500/4 = 125

| ID | span = n_local*h<br><br>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.

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

**Ejercicio 5 (20 puntos)**

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 despliéguelo en pantalla.

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

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

Recuerde agregar screenshots de la ejecución de sus programas.