

Universidad del Valle de Guatemala

Facultad de ingeniería



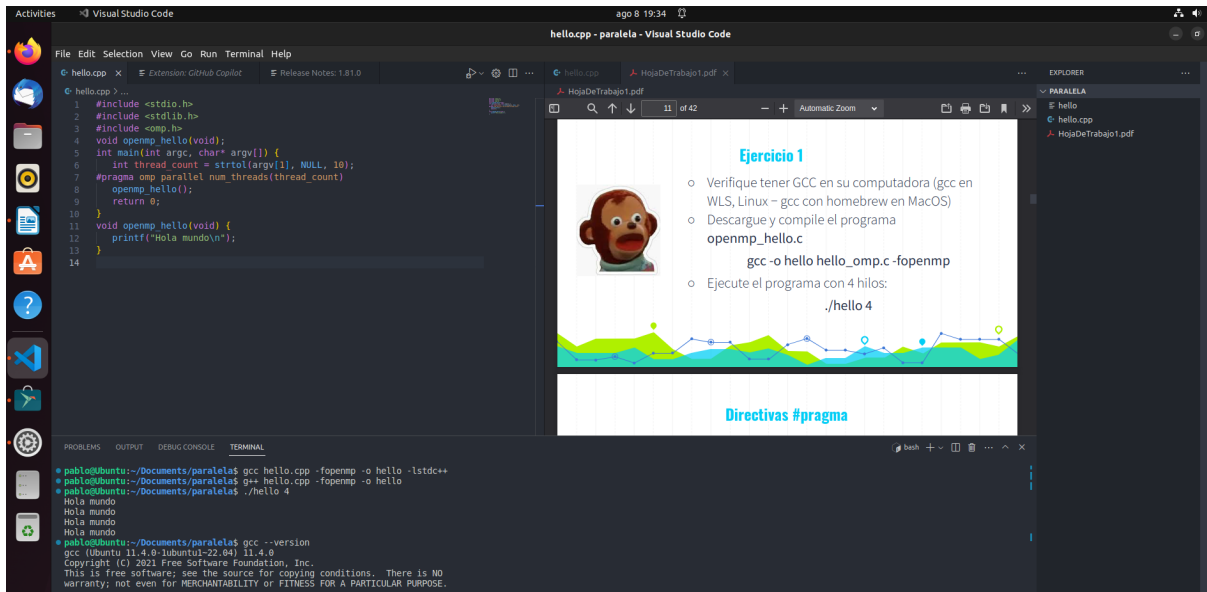
COMPUTACIÓN PARALELA Y DISTRIBUIDA - SECCIÓN - 20

Hoja de trabajo 1

Pablo Escobar 20936

Guatemala 8 de agosto del 2023

Ejercicio 1

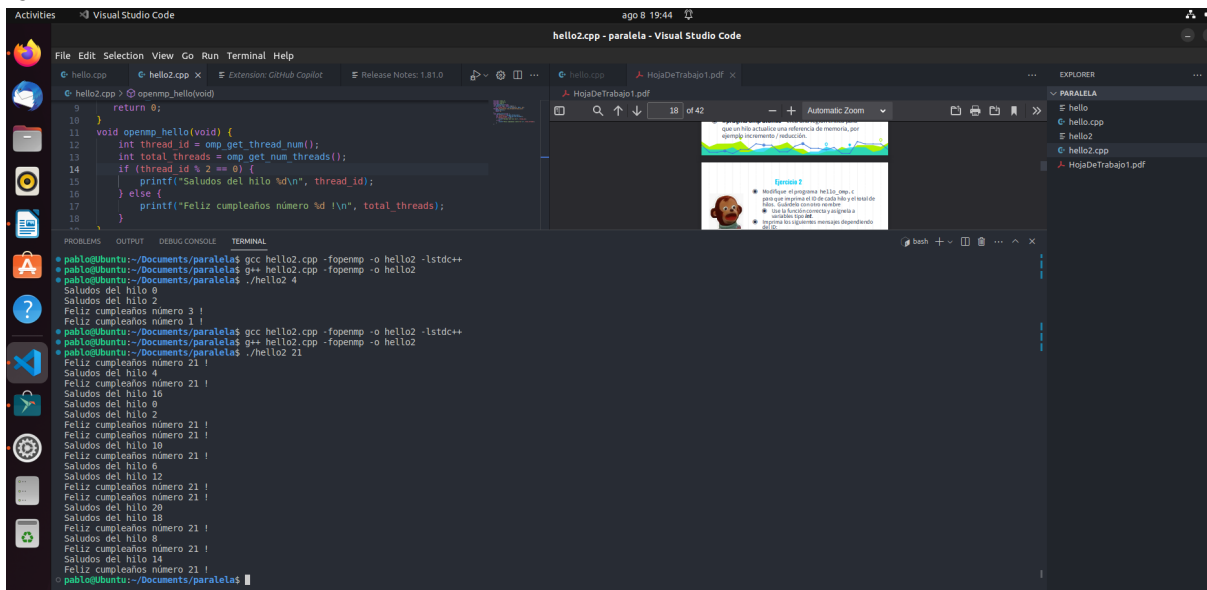


```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4 void openmp_hello(void);
5 int main(int argc, char* argv[]) {
6     int thread_count = strtol(argv[1], NULL, 10);
7     #pragma omp parallel num_threads(thread_count)
8     openmp_hello();
9     return 0;
10 }
11 void openmp_hello(void) {
12     printf("Hola mundo\n");
13 }
14
```

```
pablo@ubuntu:~/Documents/paralela$ gcc hello.cpp -fopenmp -o hello -lstdc++
pablo@ubuntu:~/Documents/paralela$ g++ hello.cpp -fopenmp -o hello
pablo@ubuntu:~/Documents/paralela$ ./hello 4
Hola mundo
Hola mundo
Hola mundo
Hola mundo

pablo@ubuntu:~/Documents/paralela$ gcc -version
gcc (Ubuntu 11.4.0-1ubuntu1-22.04) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Ejercicio 2



```
9 return 0;
10 }
11 void openmp_hello2(void) {
12     int thread_id = omp_get_thread_num();
13     int total_threads = omp_get_num_threads();
14     if (thread_id % 2 == 0) {
15         printf("Saludos del hilo %d\n", thread_id);
16     } else {
17         printf("Feliz cumpleaños número %d\n", total_threads);
18     }
19 }
20
```

```
pablo@ubuntu:~/Documents/paralela$ gcc hello2.cpp -fopenmp -o hello2 -lstdc++
pablo@ubuntu:~/Documents/paralela$ g++ hello2.cpp -fopenmp -o hello2
pablo@ubuntu:~/Documents/paralela$ ./hello2 4
Saludos del hilo 0
Saludos del hilo 2
Feliz cumpleaños número 3 !
Feliz cumpleaños número 3 !
pablo@ubuntu:~/Documents/paralela$ gcc hello2.cpp -fopenmp -o hello2 -lstdc++
pablo@ubuntu:~/Documents/paralela$ g++ hello2.cpp -fopenmp -o hello2
pablo@ubuntu:~/Documents/paralela$ ./hello2 21
Saludos del hilo 0
Feliz cumpleaños número 21 !
Saludos del hilo 4
Feliz cumpleaños número 21 !
Saludos del hilo 16
Saludos del hilo 8
Saludos del hilo 2
Feliz cumpleaños número 21 !
Feliz cumpleaños número 21 !
Saludos del hilo 10
Feliz cumpleaños número 21 !
Saludos del hilo 6
Saludos del hilo 12
Feliz cumpleaños número 21 !
Feliz cumpleaños número 21 !
Saludos del hilo 20
Saludos del hilo 18
Feliz cumpleaños número 21 !
Saludos del hilo 8
Feliz cumpleaños número 21 !
Saludos del hilo 14
Feliz cumpleaños número 21 !
pablo@ubuntu:~/Documents/paralela$
```

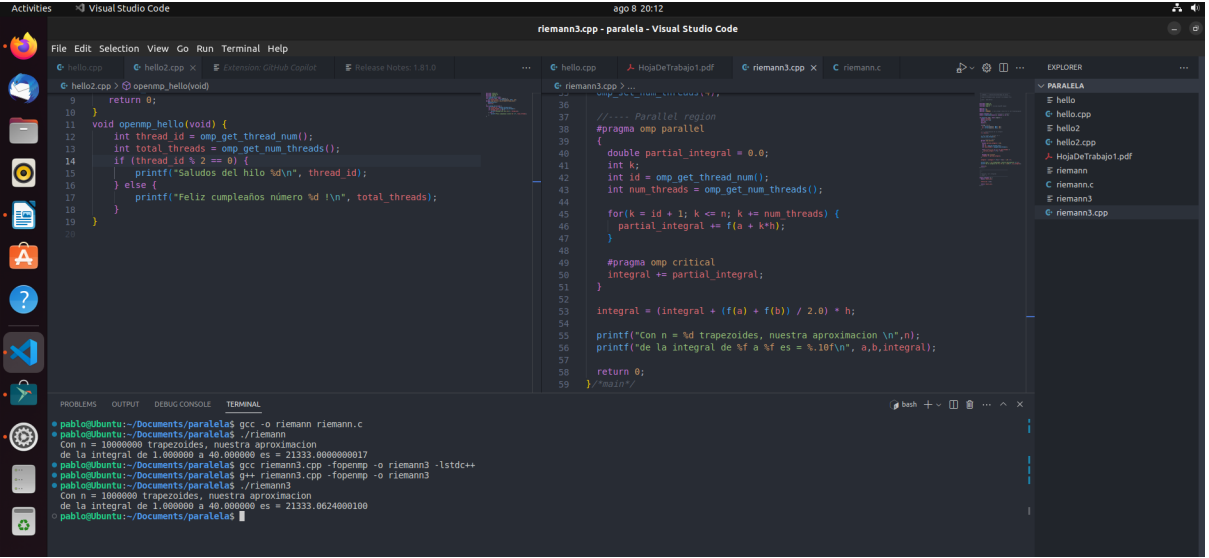
Ejercicio 3

En parejas, discuta cómo funciona el programa, el cálculo de las sumas y el paso de parámetros

Este código calcula el área bajo una curva utilizando sumas de Riemann. En la función principal "main", se definen valores iniciales para los límites de integración y el número de trapezoides que se usarán para aproximar el área. Si se proporcionan argumentos de línea de comando, estos valores pueden ser reemplazados. Luego, se llama a la función "trapezoides" para calcular la integral utilizando la regla del trapecio.

Esta función divide el intervalo de integración en trapezoides y calcula el área de cada uno. La función "f" define la función que se está integrando, en este caso, es una función cuadrática simple pero puede ser reemplazada por cualquier otra función deseada. En cuanto a los parámetros, si se proporcionan argumentos de línea de comando, estos se utilizan para reemplazar los valores iniciales.

Los argumentos se pasan a la función "main" a través de los parámetros "argc" y "argv". El parámetro "argc" indica el número de argumentos proporcionados, mientras que el parámetro "argv" es un arreglo de cadenas que contiene los argumentos en sí. Los valores de los límites de integración se obtienen convirtiendo los primeros dos argumentos a números utilizando la función "strtoul".



The screenshot shows the Visual Studio Code editor with two C++ files open: `hello2.cpp` and `riemann3.cpp`. The `riemann3.cpp` file contains the main logic for calculating the area under a curve using a parallel Riemann sum. The code includes headers for `omp` and `math`, and defines a function `f` that returns the value of a quadratic function. The `main` function takes command-line arguments and uses `omp_set_num_threads` to set the number of threads. It then calls the `trapezoides` function to calculate the integral. The `trapezoides` function is implemented using a parallel loop with `omp parallel` and `omp for` directives. The terminal at the bottom shows the execution of the program with various command-line arguments, including the number of trapezoids and the limits of integration.

```
riemann3.cpp - paralela - Visual Studio Code

36 //--- Parallel region
37 #pragma omp parallel
38 {
39     double partial_integral = 0.0;
40     int k;
41     int id = omp_get_thread_num();
42     int num_threads = omp_get_num_threads();
43     for(k = id + 1; k <= n; k += num_threads) {
44         partial_integral += f(a + k*h);
45     }
46     #pragma omp critical
47     integral += partial_integral;
48 }
49
50 integral = (integral + (f(a) + f(b)) / 2.0) * h;
51
52 printf("Con n = %d trapezoides, nuestra aproximacion\n",n);
53 printf("de la integral de %f a %f es = %.10f\n", a,b,integral);
54
55 return 0;
56 }

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

pablo@ubuntu:~/Documents/paralela$ gcc -o riemann riemann.c
pablo@ubuntu:~/Documents/paralela$ ./riemann
Con n = 1000000 trapezoides, nuestra aproximacion
de la integral de 1.000000 a 40.000000 es = 21333.00000000017
pablo@ubuntu:~/Documents/paralela$ gcc riemann3.cpp -fopenmp -o riemann3 -lstdc++
pablo@ubuntu:~/Documents/paralela$ ./riemann3
Con n = 1000000 trapezoides, nuestra aproximacion
de la integral de 1.000000 a 40.000000 es = 21333.0624000100
pablo@ubuntu:~/Documents/paralela$
```

```
File Edit Selection View Go Run Terminal Help
hello2.cpp x Extensions: GitHub Copilot Release Notes: 1.81.0
hello2.cpp > openmp_hello(void)
9 return 0;
10
11 void openmp_hello(void) {
12     int thread_id = omp_get_thread_num();
13     int total_threads = omp_get_num_threads();
14     if (thread_id % 2 == 0) {
15         printf("Saludos del hilo %d\n", thread_id);
16     } else {
17         printf("¡Feliz cumpleaños número %d!\n", total_threads);
18     }
19 }
20

riemann3.cpp > ...
12
13 #define A 1
14 #define B 40
15 #define N 1000000 // Use integer value for N, not floating-point
16
17 double f(double x); //La funcion a integrar
18 double trapezoides(double a, double b, int n);
19
20 int main(int argc, char* argv[]) {
21     double integral;
22     double a=A, b=B;
23     int n=N;
24     double h;
25     int num_threads = 4; // Default number of threads
26
27     if(argc > 1) {
28         a = strtol(argv[1], NULL, 10);
29         b = strtol(argv[2], NULL, 10);
30         if(argc > 3) {
31             num_threads = strtol(argv[3], NULL, 10);
32         }
33     }
34
35     //.... Aproximacion de la integral
36     ...
37 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
pablo@ubuntu:~/Documents/paralela$ gcc -o riemann riemann.c
pablo@ubuntu:~/Documents/paralela$ ./riemann
Con n = 1000000 trapezoides, nuestra aproximacion
de la integral de 1.000000 a 40.000000 es = 21333.00000000017
pablo@ubuntu:~/Documents/paralela$ gcc riemann3.cpp -fopenmp -o riemann3 -lstdc++
pablo@ubuntu:~/Documents/paralela$ g++ riemann3.cpp -fopenmp -o riemann3
pablo@ubuntu:~/Documents/paralela$ ./riemann3
Con n = 1000000 trapezoides, nuestra aproximacion
de la integral de 1.000000 a 40.000000 es = 21333.0624000100
pablo@ubuntu:~/Documents/paralela$ gcc riemann3.cpp -fopenmp -o riemann3 -lstdc++
pablo@ubuntu:~/Documents/paralela$ g++ riemann3.cpp -fopenmp -o riemann3
pablo@ubuntu:~/Documents/paralela$ ./riemann3 7
Segmentation fault (core dumped)
pablo@ubuntu:~/Documents/paralela$ ./riemann3 8
Segmentation fault (core dumped)
pablo@ubuntu:~/Documents/paralela$ ./riemann3 2 4 9
Con n = 1000000 trapezoides, utilizando 9 threads, nuestra aproximacion
de la integral de 2.000000 a 4.000000 es = 18.666986667
pablo@ubuntu:~/Documents/paralela$
```

Ejercicio 4

The screenshot shows a code editor on the left with C++ code for a parallel Riemann integral calculation using OpenMP. The code defines a function `trapezoides` and a `main` function that calculates the integral of $f(x) = x^2$ from 0 to 100 using 9 threads. The right side shows a presentation slide titled "Ejercicio 4" with a monkey icon. The slide lists the tasks: modify `riemann2` to include local variables (`n_local`, `a_local`, `b_local`) and include OpenMP functions to obtain the number of threads and thread ID.

```
41 // Set number of threads
42 omp_set_num_threads(num_threads);
43
44 // Parallel region
45 #pragma omp parallel
46 {
47     double partial_integral = 0.0;
48     int k;
49     int id = omp_get_thread_num();
50     int threads = omp_get_num_threads();
51
52     local_n = n / threads;
53     local_a = a + id * local_n * h;
54     local_b = local_a + local_n * h;
55
56     printf("Hilo %d de %d, calculando desde %f hasta %f\n", id, threads,
57           local_a, local_b);
58
59     for(k = id + 1; k <= n; k += threads) {
60         partial_integral += f(a + k*h);
61     }
62
63 #pragma omp critical
64 {
```

Ejercicio 4

- Modifique el programa `riemann2` para incluir los valores locales
 - `n_local`
 - `a_local`
 - `b_local`
- Recuerde incluir las funciones OpenMP para obtener:
 - Número de hilos
 - ID del hilo

Ejercicio 5

The screenshot shows a code editor on the left with C++ code for a parallel Riemann integral calculation using OpenMP. The code defines a function `trapezoides` and a `main` function that calculates the integral of $f(x) = x^2$ from 0 to 100 using 9 threads. The right side shows a presentation slide titled "Ejercicio 5" with a monkey icon. The slide lists the tasks: modify `riemann2` to add a global sum, a global sum pointer, and accumulation of `suma_local` to `suma_global`. It also includes a reminder to pass `suma_global` by reference and to protect the critical section where `suma_local` is accumulated to `suma_global`.

```
51 local_n = n / threads;
52 local_a = a + id * local_n * h;
53 local_b = local_a + local_n * h;
54
55 printf("Hilo %d de %d, calculando desde %f hasta %f\n", id, threads,
56       local_a, local_b);
57
58 local_sum = trapezoides(local_a, local_b, local_n);
59
60 #pragma omp critical
61 {
62     global_sum += local_sum;
63 }
64
65 integral = (global_sum + (f(a) + f(b)) / 2.0) * h;
66
67 printf("Con n = %d trapezoides, utilizando %d threads, nuestra aproximacion
68       (de la integral de %f a %f es = %.10f\n", a,b,integral);
69
70 printf("Suma global = %.10f\n", global_sum);
71 return 0;
72 }
73 //main
74
75 //-----
```

Ejercicio 5

- Modifique el programa `riemann2` para agregar
 - `suma_local`
 - suma global como puntero
 - acumulación de `suma_local` a `suma_global`
- Recuerde:
 - Pasar la `suma_global` como referencia (&) a la rutina de hilos
 - Proteger la sección crítica donde acumulamos la `suma_local` a la `suma_global`