

Informe del parcial

Tema: programación lineal vs programación paralela

Nota

Estudiante	Escuela	Asignatura
Rodrigo Infanzón Acosta rinfanzona@ulasalle.edu.pe	Carrera Profesional de Ingeniería de Software	Lenguaje de Programación 3

Item	Tema	Duración
Parcial	Programación secuencial vs programación paralela	3 semanas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	11/04/24	10/05/24

1. Actividades a realizar:

- Realizar la resolución y programación en el lenguaje de preferencia para resolver el problema del trapecio, tanto para la programación secuencial y programación paralela. El programa deberá empezar los cálculos para hallar el área (una integral aproximada) desde un trapecio hasta n trapecios. Deberá detenerse cuando el valor hallado se repita.
- Entrada: función cartesiana a evaluar, limite superior, limite inferior.
- Realizar mediciones de tiempo de la resolución del problema del trapecio.
- Analizar las gráficas de ambas programaciones e inferir.
- Salida: el área aproximada de la integral, trapeciosecuencial.dat, trapecioparalelo.dat.

2. Consideraciones generales del docente:

- Enviar satisfactoriamente todas sus implementaciones hacia su repositorio privado lp3-24a provenientes de las plataformas Git y GitHub.
- Describir antecedentes previos que sean necesarios para desarrollar el laboratorio. Las entregadas por el docente y/o las que se buscaron personalmente.
- Elaborar la lista de commits que permitirán culminar el laboratorio, previo a la implementación.
- Explicar porciones de código fuente importantes, trascendentales que permitieron resolver el laboratorio y que reflejen su particularidad única, sólo en trabajos grupales se permite duplicidad

- Mostrar comandos, capturas de pantalla, explicando la forma de replicar y ejecutar el entregable del laboratorio.
- Toda información externa de fuente perteneciente a otro/a autor, debe ser citada y referenciada en el bloque: referencias y referencias bibliográficas.

3. Entregables:

- URL al directorio específico del laboratorio en su repositorio GitHub privado donde esté todo el código fuente y otros que sean necesarios. Evitar la presencia de archivos: binarios, objetos, archivos temporales. Incluir archivos de especificación como: packages.json, requirements.txt o README.md.
- No olvidar que el docente debe ser siempre colaborador a su repositorio que debe ser privado. Usuario del docente: @rescobedoulasalle
- Se debe de describir sólo los commits más importantes que marcaron hitos en su trabajo, adjuntando capturas de pantalla, del commit, porciones de código fuente, evidencia de sus ejecuciones y pruebas.
- Siempre se debe explicar las imágenes (código fuente, capturas de pantalla, commits, ejecuciones, pruebas, etc.) con descripciones puntuales pero precisas.
- Agregar la estructura de directorios y archivos de su laboratorio.

4. Recursos y herramientas utilizados:

- Sistema operativo utilizado: Windows 10 pro 22H2 de 64 bits SO. 19045.4170.
- Hardware: Ryzen 5 3550H 2.10 GHz, RAM 16 GB DDR4 2400 MHz.
- Zinjal
- Git 2.44.0.
- Visual Studio Code 1.88.0.
- Cuenta de GitHub creada con el correo institucional proporcionado por la Universidad La Salle de Arequipa: rinfanzona@ulasalle.du.pe
- Conocimientos básicos en Git.
- Conocimientos intermedios en programación.
- Lenguaje de programación C++.
- Librerías y recursos en C++.
- GNUplot 5.4 patchlevel 8.
- GNU compiler 13.2.

5. URL de Repositorio Github:

- URL del Repositorio GitHub para clonar o sincronizar:
- <https://github.com/RodrigoStranger/lp3-24a>
- URL para el parcial en el Repositorio GitHub:
- <https://github.com/RodrigoStranger/lp3-24a/tree/main/parcial/>

6. Actividades previas en el repositorio de GitHub

En el desarrollo de las semanas previo al examen parcial, el profesor encargado del curso nos enseñó el manejo de ramas en Git, ya que es de vital importancia para poder realizar modificaciones a un código fuente para que cada uno pueda tener una versión actualizada del código sin interferir con el trabajo de los demás. El uso de ramas en Git permite a los desarrolladores trabajar en diferentes características o correcciones de errores de forma aislada, sin afectar el código base o el trabajo de otros miembros del equipo. Esto promueve la colaboración eficiente y reduce los conflictos que pueden surgir al trabajar en un mismo código simultáneamente.

Durante el proceso de aprendizaje, es importante comprender cómo crear, fusionar y eliminar ramas en Git, así como también cómo gestionar conflictos si surgen al fusionar ramas. Además, entender el flujo de trabajo de ramas, como trabajar en ramas de características, ramas de corrección de errores y ramas de lanzamiento, es crucial para organizar el desarrollo del proyecto de manera efectiva.

Por ello, cada uno creo su respectiva rama, en mi caso, me tocó crear la rama02 con un compañero del curso, una vez creado la rama, en el repositorio compartido por el profesor encargado del curso, en casa, podemos clonar el repositorio y situarnos en nuestra rama, con el propósito de obtener nuestro desarrollo de código en la clase, ya establecida por el profesor, podemos hacerlo de la siguiente manera:

Abrimos nuestro comando de windows o git bash y escribimos los siguientes comandos:

Listing 1: Clonar el repositorio

```
cd C:  
git clone https://github.com/rescobedoulasalle/metodo_de_trapecio
```

Luego de haber clonado el repositorio, nos situamos en la raíz de la carpeta, ejecutando el comando de windows para ejecutar las siguientes líneas:

Listing 2: Situarnos en nuestra rama elaborada

```
cd C:\metodo_de_trapecio  
git checkout rama02
```

Esto nos conducirá hacia nuestra rama y por ende, a nuestro código elaborado.



	README	28/04/2024 09:05	Archivo de origen ...	1 KB
	Trapecio	28/04/2024 09:05	Archivo de origen ...	2 KB

Imagen con propiedad de ©Rodrigo Infanzón Acosta, ©Windows

7. Programación secuencial:

La programación secuencial es uno de los paradigmas fundamentales en el desarrollo de software. Se centra en la ejecución de instrucciones en un orden específico, una tras otra, siguiendo una secuencia lógica. Este enfoque es fundamental en la construcción de algoritmos y aplicaciones donde la secuencia de las operaciones es crucial para el funcionamiento correcto del programa.

7.1. Orígenes de la programación secuencial:

La programación secuencial ha estado presente desde los primeros días de la informática moderna. Su origen se remonta a los primeros lenguajes de programación y sistemas informáticos en las décadas de 1950 y 1960. A medida que la informática evolucionaba, la programación secuencial se convirtió en el método estándar para la mayoría de las aplicaciones de software. El concepto de secuencia de instrucciones es fundamental en la arquitectura de las primeras computadoras, donde las operaciones se ejecutaban secuencialmente, una después de la otra. Lenguajes de programación como Fortran, Cobol y Assembly proporcionaron a los programadores las herramientas necesarias para expresar secuencias de instrucciones de manera más legible y estructurada.

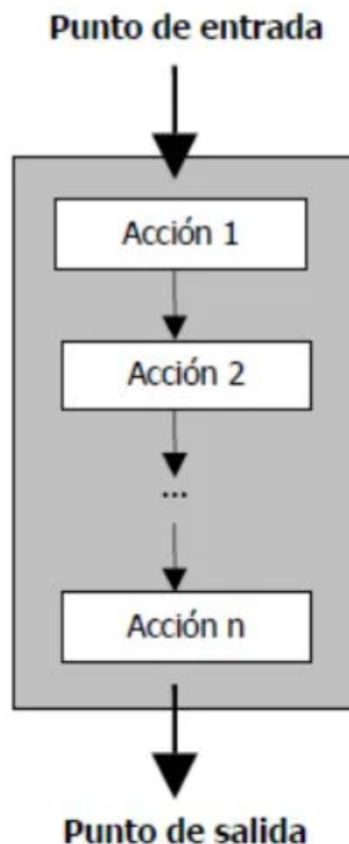


Imagen con propiedad de ©Issuu

7.2. Características de la programación secuencial:

- 1.- **Secuencia de Instrucciones:** las instrucciones se ejecutan secuencialmente, una después de la otra, en el orden en que están escritas en el código.
- 2.- **Flujo de Control Lineal:** el flujo de control sigue una trayectoria lineal a través del código, lo que significa que las instrucciones se ejecutan en el orden en que aparecen, sin bifurcaciones ni repeticiones.
- 3.- **Ejecución Determinista:** dado un conjunto de datos de entrada particular, la ejecución de un programa secuencial siempre producirá el mismo resultado, ya que las instrucciones se ejecutan en el mismo orden cada vez.

7.3. Ventajas de la programación secuencial:

- 1.- **Simplicidad y Claridad:** la estructura lineal de la programación secuencial hace que sea fácil de entender y seguir, lo que la convierte en una opción ideal para desarrolladores principiantes y para proyectos con requisitos simples. La secuencia de instrucciones proporciona una lógica directa y fácil de seguir.
- 2.- **Facilidad de Depuración:** debido a que las instrucciones se ejecutan en un orden predecible, la identificación y corrección de errores en el código es relativamente sencilla. Los desarrolladores pueden seguir el flujo de ejecución de manera lineal, lo que facilita la localización de problemas y su solución.
- 3.- **Control Determinista:** la ejecución de un programa secuencial produce resultados consistentes y predecibles para un conjunto de datos de entrada dado. Esto facilita la validación y verificación del comportamiento del programa, lo que es crucial para aplicaciones críticas donde la precisión es esencial.
- 4.- **Bajo Overhead de Recursos:** la programación secuencial tiende a tener un bajo overhead en términos de recursos computacionales, lo que la hace eficiente para procesos simples y de baja complejidad. No hay necesidad de gestionar múltiples hilos de ejecución ni de coordinar operaciones concurrentes, lo que puede reducir la carga en el sistema.

7.4. Desventajas de la programación secuencial:

- 1.- **Limitaciones en la Resolución de Problemas Complejos:** si bien la programación secuencial es efectiva para tareas simples y lineales, puede resultar limitada para problemas que requieren lógica compleja, ramificaciones condicionales o procesamiento paralelo. Para abordar estos problemas, a menudo se necesitan paradigmas de programación más avanzados, como la programación concurrente o la programación orientada a objetos.
- 2.- **Ineficiencia en Operaciones Paralelas:** en entornos donde se pueden realizar múltiples tareas simultáneamente, como sistemas con múltiples núcleos de procesamiento, la programación secuencial puede ser ineficiente. No aprovecha al máximo la capacidad de ejecutar operaciones en paralelo, lo que puede resultar en un rendimiento subóptimo del sistema.
- 3.- **Complejidad en el Mantenimiento de Código Escalable:** a medida que los proyectos de software crecen en tamaño y complejidad, mantener un enfoque estrictamente secuencial puede volverse complicado. El código puede volverse difícil de mantener y modificar, especialmente si no se sigue una estructura clara y modular. En tales casos, puede ser necesario refactorizar el código para hacerlo más modular y extensible.
- 4.- **Dificultad en la Gestión de Excepciones y Errores:** la programación secuencial puede enfrentar desafíos en la gestión de excepciones y errores, especialmente en sistemas complejos donde múltiples partes del código pueden interactuar entre sí. Es crucial implementar mecanismos robustos de manejo de errores para garantizar que el programa pueda recuperarse de manera adecuada en caso de fallos inesperados.

Aunque la programación secuencial ofrece simplicidad y claridad en el desarrollo de software, también presenta desafíos en la resolución de problemas complejos y en la eficiencia en entornos de procesamiento paralelo. Los desarrolladores deben evaluar cuidadosamente las necesidades del proyecto y considerar otros paradigmas de programación cuando sea necesario para abordar estas limitaciones.

8. Programación paralela:

La programación paralela es un paradigma de programación en el que múltiples tareas se ejecutan simultáneamente, con el objetivo de mejorar el rendimiento y la eficiencia de los sistemas informáticos. Históricamente, los procesadores mejoraron en velocidad principalmente a través del aumento de la frecuencia de reloj. Sin embargo, esta estrategia llegó a su límite debido a limitaciones físicas y de consumo de energía. Como alternativa, la programación paralela aprovecha la capacidad de los sistemas para ejecutar múltiples tareas simultáneamente, ya sea en múltiples núcleos de CPU o en sistemas distribuidos.

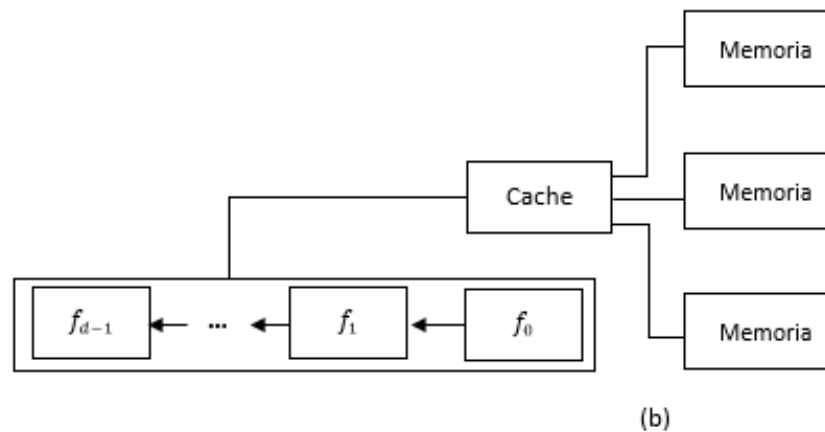


Imagen con propiedad de ©procesamientoparalelo.blogspot

Las diferentes posibilidades existentes para desarrollar sistemas paralelos hacen que una clasificación definitiva sea complicada. Se muestra una clasificación clásica propuesta por Flynn, que se basa en el ciclo de instrucciones y en el flujo de dato.

8.1. Orígenes de la programación paralela:

Los orígenes de la programación paralela se remontan al desarrollo de la computación y la necesidad de mejorar el rendimiento de los sistemas informáticos. A lo largo de la historia de la informática, varios hitos y avances tecnológicos han contribuido al desarrollo y la evolución de la programación paralela, tales como:

- 1.- Inicio de la computación:** en los primeros días de la computación, las máquinas eran sistemas monolíticos que ejecutaban una sola tarea a la vez. Estos sistemas utilizaban un solo procesador y no tenían capacidad para ejecutar múltiples tareas simultáneamente.
- 2.- Multiprogramación:** con la evolución de la tecnología, se introdujo la multiprogramación, que permitía ejecutar múltiples programas en un solo sistema mediante la alternancia rápida entre ellos. Aunque no implicaba verdadero paralelismo, sentó las bases para el desarrollo de sistemas informáticos más avanzados.
- 3.- Supercomputadoras:** en la década de 1960, se construyeron las primeras supercomputadoras,

que fueron diseñadas para realizar cálculos complejos y procesar grandes cantidades de datos. Estos sistemas utilizaban técnicas de paralelismo a nivel de instrucción para mejorar el rendimiento.

4.- Arquitecturas SIMD y MIMD: a finales de la década de 1960 y principios de la década de 1970, se desarrollaron arquitecturas de computadoras que permitían la ejecución simultánea de múltiples instrucciones (SIMD) y múltiples procesadores (MIMD). Estas arquitecturas sentaron las bases para la programación paralela moderna.

5.- Primeras herramientas de programación paralela: en la década de 1980, se introdujeron las primeras herramientas y lenguajes de programación diseñados específicamente para la programación paralela. Ejemplos incluyen el lenguaje de programación Parallel Pascal y la biblioteca de programación paralela PVM (Parallel Virtual Machine).

6.- Avances en hardware: con el avance de la tecnología de semiconductores, se hicieron posibles arquitecturas de computadoras más avanzadas, como los procesadores multinúcleo y las unidades de procesamiento gráfico (GPU). Estos avances permitieron una mayor paralelización a nivel de hardware y abrieron nuevas oportunidades para la programación paralela.

7.- Evolución de estándares y bibliotecas: a lo largo de las últimas décadas, se han desarrollado estándares y bibliotecas de programación paralela ampliamente utilizados, como OpenMP (Open Multi-Processing) y MPI (Message Passing Interface). Estas herramientas han facilitado la escritura de software paralelo y han contribuido al crecimiento de la programación paralela en una variedad de áreas de aplicación.

8.2. Características de la programación paralela:

1.- División de Tareas: en la programación paralela, las tareas se dividen en partes más pequeñas que pueden ejecutarse simultáneamente en múltiples núcleos de procesamiento.

2.- Concurrencia: múltiples tareas pueden ejecutarse simultáneamente, lo que puede mejorar significativamente el rendimiento y la eficiencia del sistema.

3.- Comunicación: los procesos o hilos en paralelo pueden necesitar comunicarse entre sí para compartir datos o coordinar sus actividades.

4.- Coordinación: es importante coordinar las actividades de los procesos paralelos para evitar condiciones de carrera y garantizar la coherencia de los datos.

5.- Escalabilidad: la programación paralela permite escalar el rendimiento de una aplicación agregando más recursos de hardware, como núcleos de CPU o nodos de procesamiento.

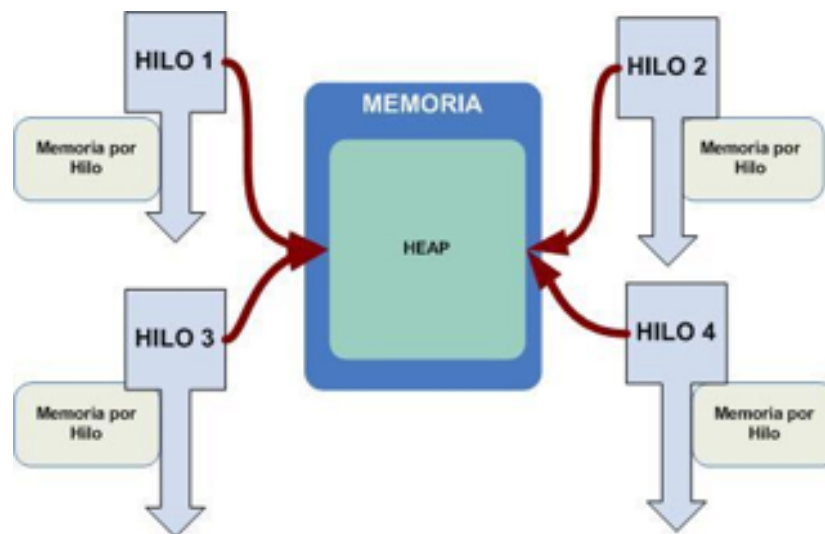


Imagen con propiedad de ©fERESTREPOCA

8.3. Ventajas de la programación paralela:

- 1.- **Mejor rendimiento:** la programación paralela puede mejorar significativamente el rendimiento de una aplicación al distribuir la carga de trabajo entre múltiples núcleos de procesamiento.
- 2.- **Mayor eficiencia:** al aprovechar la capacidad de procesamiento paralelo, las aplicaciones pueden realizar tareas de manera más eficiente y completarlas en menos tiempo.
- 3.- **Escalabilidad:** las aplicaciones diseñadas con programación paralela son más escalables, lo que significa que pueden manejar cargas de trabajo más grandes al agregar más recursos de hardware.
- 4.- **Mejor utilización de recursos:** al distribuir las tareas entre múltiples núcleos de procesamiento, se utiliza de manera más eficiente la capacidad de procesamiento disponible.

8.4. Desventajas de la programación paralela:

- 1.- **Complejidad:** la programación paralela es inherentemente más compleja que la programación secuencial, ya que requiere coordinar múltiples procesos o hilos y manejar problemas como condiciones de carrera y sincronización.
- 2.- **Dificultad de depuración:** depurar aplicaciones paralelas puede ser más difícil debido a la naturaleza concurrente de la ejecución. Los errores pueden ser difíciles de reproducir y diagnosticar.
- 3.- **Overhead de comunicación:** la comunicación entre procesos paralelos puede introducir overhead, especialmente en sistemas distribuidos, lo que puede afectar negativamente el rendimiento.
- 4.- **Escalabilidad limitada:** aunque la programación paralela ofrece escalabilidad, esta puede verse limitada por factores como la comunicación entre procesos y la naturaleza de las tareas paralelizables.
- 5.- **Sincronización y coherencia de datos:** mantener la consistencia de los datos compartidos entre procesos paralelos puede ser complicado y puede requerir técnicas avanzadas de sincronización y gestión de la coherencia de la memoria.

9. Método del trapecio: resolución general

El método del trapecio es una técnica de integración numérica utilizada para aproximar el valor de una integral definida. Su nombre proviene de la forma de los trapecios que se utilizan para aproximar el área bajo una curva. Este método es especialmente útil cuando la función que se desea integrar es complicada o no tiene una antiderivada fácilmente expresable.

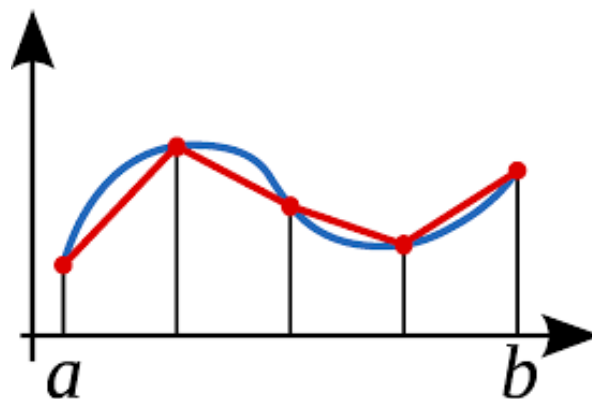


Imagen con propiedad de ©Antonio Cedillo Hernandez

La idea básica detrás del método del trapecio es dividir el área bajo la curva en una serie de trapecios más pequeños y luego sumar el área de cada trapecio para obtener una estimación de la integral total.

Cuanto más pequeños sean los trapecios, más precisa será la aproximación.
Con esta idea, podemos desarrollar una lógica entendiendo que vamos a hacer iteraciones infinitas hasta poder determinar el área de la integral aproximada:

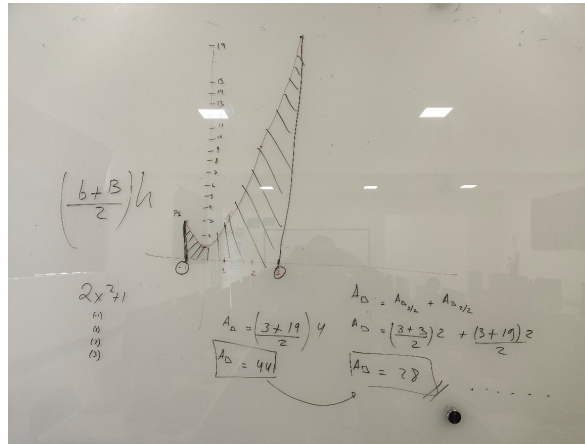


Imagen con propiedad de ©Richart Escobedo

El programa deberá empezar los cálculos para hallar el área (la integral aproximada) desde $n=1$ trapecios hasta (n) trapecios. El programa deberá detenerse cuando el valor hallado $(n-1)$ y n trapecios sean iguales o parecidos.

9.1. Desarrollo del programa Trapecioineal: programacion secuencial

La lógica general del programa es poder usar el método del trapecio para calcular aproximadamente el área bajo una curva dada una función y los límites de integración dados. El número de trapecios se incrementa gradualmente hasta que la diferencia entre las áreas calculadas con un número de trapecios y el número anterior sea menor que la precisión deseada. Una vez que se alcanza la precisión deseada, se muestra el resultado y se genera un archivo con los datos de los tiempos y áreas calculadas.

Mi logica posee:

1.- Variables globales: se definen tres variables globales:

- 1.1.- **contadordetrapecios:** Utilizada para contar el número de trapecios utilizados en el cálculo.
- 1.2.- **areatotalverdadera:** Almacena el área total calculada entre los límites dados.
- 1.3.- **precision:** Define la precisión deseada para el cálculo.

2.- Clase Trapecio: se define una clase Trapecio para representar un trapecio. Tiene tres atributos privados (altura, basemayor, basemenor) y un constructor para inicializar esos atributos. También tiene un método público **calculararea()** para calcular el área del trapecio.

3.- Función y(double x): define una función $y(x)$ que devuelve el valor de la función a integrar en un punto x .

4.- Función generararchivo(vector<double> tiempos, vector<double> areas): esta función recibe dos vectores: tiempos y areas. Se encarga de escribir los datos en un archivo llamado "TrapecioSecuencial.dat".

5.- Función areatotal(double limiteinferior, double limitesuperior, int precision): esta es la función principal para calcular el área total. Utiliza un bucle while para calcular el área con una precisión específica. Incrementa el número de trapecios en cada iteración hasta que la diferencia entre el área calculada con un número de trapecios y el área calculada con un número menor de trapecios sea menor que la tolerancia definida por la precisión.

6.- Función saludarsegunhora(): esta función muestra un saludo según la hora del día en que se ejecuta el programa.

7.- Función main(): la función principal del programa. Inicia saludando al usuario según la hora del día. Luego, entra en un bucle donde solicita al usuario los límites de integración y la precisión deseada. Si los límites son válidos, se inicia el cálculo del área total utilizando la función areatotal(). Finalmente, el programa termina cuando se completa el cálculo del área total.

9.1.1. Diagrama de flujo del método secuencial:

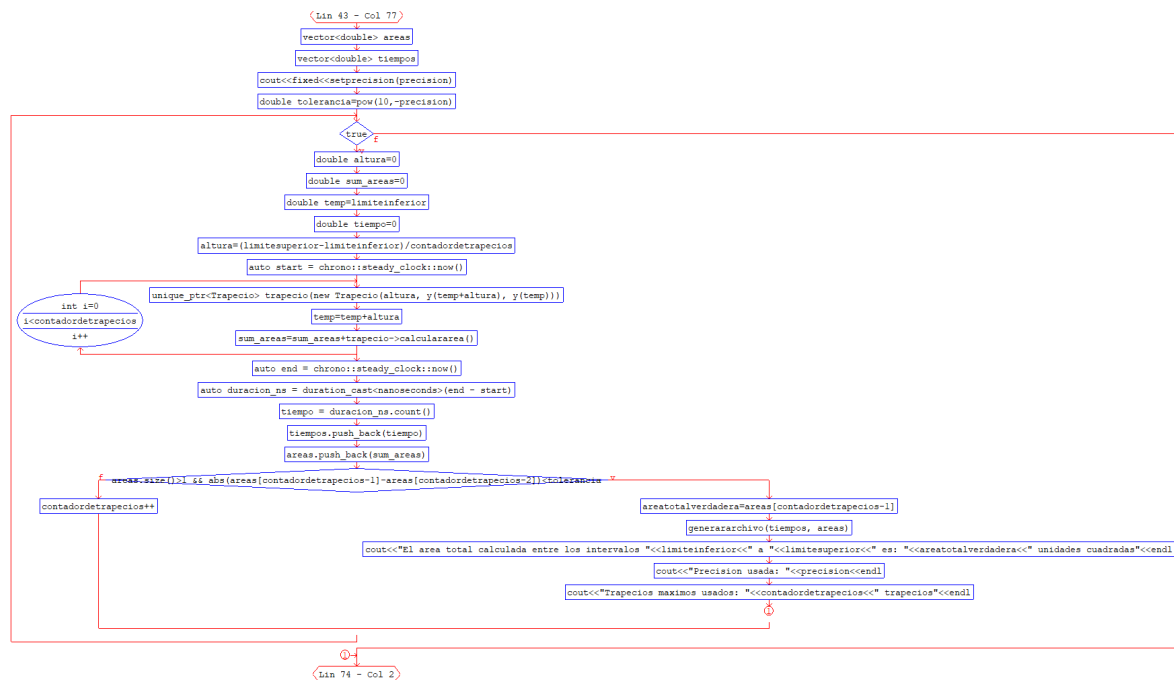


Imagen con propiedad de ©Rodrigo Infanzon, ©Zinjal

9.1.2. Commits importantes del código:

Commit 4: 464348fbba45b22d51380c9f8774c23528ce0c39

La función areatotal ha sido mejorada significativamente. Se ha simplificado el cálculo del área total utilizando una fórmula directa. Además, se maneja la precisión de los resultados según la cantidad de dígitos decimales especificados por el usuario, garantizando resultados precisos. Se registra el tiempo de cálculo para cada iteración, lo que permite un seguimiento detallado del rendimiento del algoritmo. Estas mejoras hacen que la función sea más eficiente, clara y fácil de mantener.

Commit 8: f49e2bd95bee0156396a5696c136aca826a24ff7

En la función main, se solicitan al usuario tres parámetros: el límite inferior y superior del intervalo sobre el cual se calculará la integral definida, y la precisión deseada, que representa el número de dígitos decimales para el resultado. Una vez que el usuario proporciona estos valores, se llama a la función calcularAreaTotal, pasando los parámetros ingresados. La función calcularAreaTotal, que ahora está definida como void, se encarga de realizar el cálculo del área total utilizando el método del trapecio y almacenar el tiempo de ejecución. Esta separación permite un diseño más modular y facilita la medición del rendimiento del algoritmo.

Commit 9: 0498a3c4e0372a6882cfff83311f09091a5306a4

En este commit, se presenta la explicación del programa por medio de comentarios, esto ayuda a com-

prender en un primer momento la estructura del código, como se implementó, y la lógica estructurada que posee.

Commit final : 058a45707c82def339e59c0da9c1e9f3f498d796

En este último commit se presisa los decimales a 6, se mejora la implementación de las funciones, limpieza general. Se agregó también comentarios para entender la lógica del código

9.1.3. Código fuente: programación secuencial, método del trapecio

Listing 3: Código fuente: método del trapecio en programación secuencial

```
1 #include <iostream> // Incluye la librería de entrada y salida estándar
2 #include <iomanip> // Incluye la librería para manipulación de entrada/salida
3 #include <cmath> // Incluye la librería matemática
4 #include <vector> // Incluye la librería de vectores
5 #include <fstream> // Incluye la librería para manejo de archivos
6 #include <ctime> // Incluye la librería para funciones de tiempo
7 #include <chrono> // Incluye la librería para medición de tiempo
8 #include <memory> // Incluye la librería para manejo de punteros inteligentes
9
10 using namespace std; // Utiliza el espacio de nombres estándar
11 using namespace chrono; // Utiliza el espacio de nombres para funciones de tiempo
12
13 int contadordeTrapecios = 1; // Contador global para el número de trapecios
14 string n_archivo = "TrapecioSecuencial.dat"; // Nombre del archivo de salida
15
16 // Definición de la clase Trapecio
17 class Trapecio {
18 private:
19     double altura; // Altura del trapecio
20     double basemayor; // Base mayor del trapecio
21     double basemenor; // Base menor del trapecio
22 public:
23     // Constructor que inicializa las dimensiones del trapecio
24     Trapecio(double _altura, double _basemayor, double _basemenor): altura(_altura),
        basemayor(_basemayor), basemenor(_basemenor) {}
25
26     // Método que calcula el área del trapecio
27     double calculararea() {
28         return ((basemayor + basemenor) * altura) / 2; // Fórmula para el área del trapecio
29     }
30 };
31
32 // Función que define la ecuación a integrar
33 double y(double x) {
34     return 2 * pow(x, 2) + 1; // Retorna el valor de la función en x
35 }
36
37 // Función para generar un archivo con los tiempos y áreas calculadas
38 void generararchivo(vector<double> tiempos, vector<double> areas) {
39     ofstream myfile(n_archivo); // Abre el archivo para escribir
40     int size_t = tiempos.size(); // Obtiene el tamaño del vector de tiempos
41     for(int i = 0; i < size_t; i++) { // Itera sobre los elementos del vector
42         cout << "Con " << i + 1 << " trapecios, el área calculada es: " << areas[i] << endl;
43         // Imprime el área calculada
44         if(i != size_t - 1) {
```

```
44         myfile << i + 1 << "\t" << tiempos[i] << endl; // Escribe el nmero de trapecios y
           el tiempo en el archivo
45     } else {
46         myfile << i + 1 << "\t" << tiempos[i]; // ltima linea sin salto de linea
47     }
48 }
49 myfile.close(); // Cierra el archivo
50 cout << endl;
51 }
52
53 // Funcin para calcular el rea usando n trapecios
54 double calculartrapecio_n(double contadordetrapecios, double altura, double &temp, double
    sum_areas, double limiteinferior, double limitesuperior) {
55     for (int i = 0; i < contadordetrapecios; i++) { // Itera sobre el nmero de trapecios
56         unique_ptr<Trapecio> trapecio(new Trapecio(altura, y(temp + altura), y(temp))); //
           Crea un nuevo objeto Trapecio
57         double area_actual = trapecio->calculararea(); // Calcula el rea del trapecio actual
58         sum_areas = sum_areas + area_actual; // Suma el rea actual al total
59         temp = temp + altura; // Incrementa el valor de temp
60     }
61     return sum_areas; // Retorna la suma de las reas
62 }
63
64 // Funcin para agregar datos a los vectores de tiempos y reas
65 void pusheardatos(vector<double>& tiempos, double tiempo, vector<double>& areas, double
    sum_areas) {
66     tiempos.push_back(tiempo); // Agrega el tiempo al vector de tiempos
67     areas.push_back(sum_areas); // Agrega el rea al vector de reas
68 }
69
70 // Funcin para calcular el rea total bajo la curva
71 void areatotal(double limiteinferior, double limitesuperior) {
72     vector<double> areas; // Vector para almacenar las reas calculadas
73     vector<double> tiempos; // Vector para almacenar los tiempos de clculo
74     double tolerancia = pow(10, -6); // Tolerancia para la comparacin de reas
75     while(true) { // Bucle infinito hasta que se cumpla la condicin de parada
76         double altura = 0; // Inicializa la altura
77         double sum_areas = 0; // Inicializa la suma de reas
78         double temp = limiteinferior; // Inicializa temp al lmite inferior
79         altura = (limitesuperior - limiteinferior) / contadordetrapecios; // Calcula la
           altura de cada trapecio
80
81         auto start = chrono::steady_clock::now(); // Marca el inicio del tiempo
82         sum_areas = calculartrapecio_n(contadordetrapecios, altura, temp, sum_areas,
           limiteinferior, limitesuperior); // Calcula la suma de las reas
83         auto end = chrono::steady_clock::now(); // Marca el fin del tiempo
84
85         auto duracion_ns = duration_cast<nanoseconds>(end - start).count(); // Calcula la
           duracin en nanosegundos
86         pusheardatos(tiempos, duracion_ns, areas, sum_areas); // Agrega los datos a los
           vectores de tiempos y reas
87
88         if(areas.size() > 1 && abs(areas[contadordetrapecios - 1] - areas[contadordetrapecios
           - 2]) < tolerancia) { // Verifica la condicin de tolerancia
89             generararchivo(tiempos, areas); // Genera el archivo con los datos
```

```
90     cout << "El rea total calculada entre los intervalos " << limiteinferior << " a "
        << limitesuperior << " es aproximadamente: " << areas[contadordetrapecios -
91     1] << " unidades cuadradas" << endl; // Imprime el resultado final
    cout << "Trapecios mximos usados: " << contadordetrapecios << " trapecios" <<
        endl; // Imprime el nmero de trapecios usados
92     cout << "Decimales trabajados: " << 6 << endl; // Imprime la precisin en decimales
93     cout << "Tolerancia usada: " << tolerancia << endl; // Imprime la tolerancia usada
94     cout << endl;
95     cout << "Archivo " << n_archivo << " creado" << endl; // Informa que el archivo
        fue creado
96     break; // Rompe el bucle
97 }
98
99     contadordetrapecios++; // Incrementa el contador de trapecios
100 }
101 }
102
103 // Funcin para mostrar un saludo segn la hora del da
104 void saludarsegunhora() {
105     time_t ahora = time(0); // Obtiene el tiempo actual
106     tm *horaLocal = localtime(&ahora); // Convierte el tiempo a la hora local
107     int hora = horaLocal->tm_hour; // Obtiene la hora actual
108     if (hora >= 6 && hora < 12) {
109         cout << "Buenos das, al sistema para hallar la integral definida aproximada con el
            mtodo del trapecio" << endl; // Saludo de buenos das
110     } else if (hora >= 12 && hora < 18) {
111         cout << "Buenas tardes, al sistema para hallar la integral definida aproximada con el
            mtodo del trapecio" << endl; // Saludo de buenas tardes
112     } else {
113         cout << "Buenas noches, al sistema para hallar la integral definida aproximada con el
            mtodo del trapecio" << endl; // Saludo de buenas noches
114     }
115 }
116
117 // Manipulador de flujo para establecer la precisin a 6 decimales
118 ostream& precision_six(ostream& os) {
119     os << fixed << setprecision(6); // Establece la precisin a 6 decimales
120     return os; // Retorna el flujo
121 }
122
123 // Funcin principal
124 int main() {
125     cout << precision_six; // Aplica el manipulador de flujo para precisin
126     saludarsegunhora(); // Llama a la funcin para mostrar el saludo
127     while(true) { // Bucle infinito hasta que se cumpla la condicin de salida
128         double limitesuperior; // Variable para el lmite superior
129         double limiteinferior; // Variable para el lmite inferior
130         cout << "Digite el lmite inferior: ";
131         cin >> limiteinferior; // Lee el lmite inferior
132         cout << "Digite el lmite superior: ";
133         cin >> limitesuperior; // Lee el lmite superior
134         cout << endl;
135         if(limitesuperior == limiteinferior) { // Verifica si los lmites son iguales
136             cout << "El rea total calculada entre los intervalos " << limiteinferior << " a "
                << limitesuperior << " es aproximadamente: " << 0 << " unidades cuadradas" <<
                    endl; // Imprime que el rea es cero
```

```
137         break; // Rompe el bucle
138     }
139     if(limite superior > limite inferior) { // Verifica si el lmite superior es mayor que
        el inferior
140         cout << "Calculando....." << endl; // Imprime mensaje de clculo
141         areatotal(limite inferior, limite superior); // Llama a la funcin para calcular el
            rea total
142         cout << endl;
143         break; // Rompe el bucle
144     } else {
145         cout << "Valores incorrectos o no lgicos." << endl; // Imprime mensaje de error
146         cout << "El lmite superior debe ser mayor que el lmite inferior." << endl; //
            Indica el error en los valores
147         cout << endl;
148         cout << "Ingrese de nuevo los parmetros correctos:" << endl; // Pide ingresar
            nuevamente los valores
149     }
150 }
151 return 0; // Retorna 0 indicando que el programa termin correctamente
152 }
```

9.1.4. Ejecución del algoritmo secuencial:

Para la ejecución del algoritmo, en nuestro entorno de desarrollo preferido, o en consola, podemos ejecutar el programa de la siguiente manera:

Listing 4: Ejecutar Trapeciolineal.cpp

```
1 C:\lp3-24a\parcial\exercises
2 g++ -o Trapeciolineal.exe Trapeciolineal.cpp
3 Trapeciolineal.exe
```

Una vez ejecutado el programa, nos aparecerá la ventana de consola en donde nosotros debemos introducir los siguientes datos:

Listing 5: Interacción:

```
1 Buenas noches, al sistema para hallar la integral definida aproximada con el metodo del
   trapecio
2 Digite el limite inferior: 0
3 Digite el limite superior: 10
```

Esto nos dara el resultado final, conjuntamente creandonos un archivo TrapecioSecuencial.dat, este archivo lo podemos encontrar en el output de nuestro entorno de desarrollo, o si estamos desarrollando online, en la ventana vecina.

Listing 6: Salida del programa:

```
1 Calculando.....
2 .....
3 .....
4 .....
5 Con 395 trapecios, el area calculada es: 676.66880
6 Con 396 trapecios, el area calculada es: 676.66879
7 Con 397 trapecios, el area calculada es: 676.66878
8 Con 398 trapecios, el area calculada es: 676.66877
```

```
9      Con 399 trapecios, el area calculada es: 676.66876
10     Con 400 trapecios, el area calculada es: 676.66875
11     Con 401 trapecios, el area calculada es: 676.66874
12     Con 402 trapecios, el area calculada es: 676.66873
13     Con 403 trapecios, el area calculada es: 676.66872
14     Con 404 trapecios, el area calculada es: 676.66871
15     Con 405 trapecios, el area calculada es: 676.66870
16     Con 406 trapecios, el area calculada es: 676.66869
17
18     El area total calculada entre los intervalos 0.00000 a 10.00000 es: 676.66869 unidades
        cuadradas
19     Trapecios maximos usados: 406 trapecios
20
21 << El programa ha finalizado: codigo de salida: 0 >>
22 << Presione enter para cerrar esta ventana >>
```

Observamos que el programa nos devuelve el area aproximada calculada a partir de la función ya definida en el codigo, en la cual tambien se puede modificar, pero interamente, por otro lado, nos muestra los trapecios utilizados, y la precision usada.

9.2. Desarrollo del programa Trapecioparalelo: programacion paralela

El cálculo del área total se realiza en paralelo, lo que implica dividir el intervalo de integración en varios subintervalos, cada uno representando un trapecio. Para gestionar este proceso, el código utiliza la biblioteca thread, permitiendo la creación y gestión de múltiples hilos de ejecución.

Variables Relevantes:

n-threads: Esta variable determina el número de hilos que se utilizarán para realizar el cálculo en paralelo.

contadordetrapecios: Lleva la cuenta del número de trapecios utilizados para la aproximación.

n-archivo: Almacena el nombre del archivo donde se registrarán los datos de tiempo y áreas calculadas.

decimales: Indica el número de decimales a utilizar en los cálculos.

mutex: Se utiliza para garantizar la consistencia en el acceso a variables compartidas entre los hilos, como contadordetrapecios. Proceso de Cálculo: el intervalo de integración se divide en múltiples subintervalos, cada uno asignado a un hilo. Cada hilo calcula el área de los trapecios en su subintervalo utilizando el método del trapecio. Aquí, la clase Trapecio y la función $y(x)$ definen la forma de cada trapecio y la función a integrar, respectivamente.

La variable sum-areas se utiliza para acumular las áreas calculadas por cada hilo, garantizando su consistencia mediante el uso de un mutex. Una vez que se han calculado todas las áreas parciales, se suman para obtener el área total aproximada bajo la curva. Rendimiento y Análisis:

El código incluye funciones para medir el tiempo de ejecución y guardar los resultados en un archivo para su posterior análisis. Esto permite evaluar el rendimiento del cálculo en paralelo en función del número de trapecios utilizados en la aproximación.

9.2.1. Commits importantes del código:

Commit 6: **ac52526efd016f31b9eebe221ee37cab4b82759c**

Se han agregado funciones y correcciones al código para mejorar su modularidad, rendimiento y robustez. En primer lugar, se introdujo la función $y(x)$ para definir la función que se integrará utilizando

el método del trapecio. Esto mejora la claridad del código al separar la definición de la función matemática del resto de la lógica del programa. Además, se creó la clase Trapecio para encapsular la lógica relacionada con el cálculo del área de un trapecio, lo que facilita su reutilización y mantenimiento.

En cuanto a las correcciones, se utilizó una variable atómica `totalArea` para garantizar la consistencia al acumular las áreas calculadas por cada hilo en el cálculo en paralelo del área total. También se realizaron ajustes en la función `areatotalparalela()` para dividir adecuadamente el intervalo de integración entre los hilos y para gestionar la finalización de los hilos de manera adecuada. Además, se mejoró la función `medirTiempos()` para escribir los resultados en un archivo de texto, lo que facilita el análisis de los tiempos de ejecución para diferentes cantidades de trapecios. Estas adiciones y correcciones enriquecen el código al hacerlo más claro, eficiente y fácil de mantener.

Commit final: 06460d09f8101f57ec88a3c0735d74a2d16d0b28

En este último commit, se reestructuró el algoritmo para mejorar su capacidad de procesamiento paralelo. Se implementó un enfoque más eficiente al dividir el intervalo de integración en subintervalos y distribuir el cálculo entre varios hilos de ejecución. La reestructuración se centró en el bucle principal, donde se inicializa la suma de áreas, se calcula el intervalo que cada hilo procesará y se preparan los hilos para su ejecución. Utilizando un bucle `for`, se asigna a cada hilo una porción del intervalo y se define una función lambda que calcula las áreas locales de los trapecios en ese subintervalo. Cada hilo realiza el cálculo de forma independiente, y los resultados parciales se almacenan en un array compartido `sum_areas_hilos`, asegurando la consistencia con el uso de un `mutex`. Esta reestructuración mejora la eficiencia del algoritmo al

Listing 7: Código fuente: método del trapecio en programación paralela

```
1 #include <iostream> // Incluye la librería de entrada y salida estandar
2 #include <iomanip> // Incluye la librería para manipulacin de entrada/salida
3 #include <cmath> // Incluye la librería matemática
4 #include <vector> // Incluye la librería de vectores
5 #include <fstream> // Incluye la librería para manejo de archivos
6 #include <ctime> // Incluye la librería para funciones de tiempo
7 #include <chrono> // Incluye la librería para medicion de tiempo
8 #include <memory> // Incluye la librería para manejo de punteros inteligentes
9 #include <thread> // Incluye la librería para manejo de hilos
10 #include <mutex> // Incluye la librería para manejo de mutex
11
12 int n_threads = 2; // Nmero de hilos a usar
13
14 using namespace std; // Utiliza el espacio de nombres estandar
15 using namespace chrono; // Utiliza el espacio de nombres para funciones de tiempo
16
17 mutex mtx; // Define un mutex para proteger secciones crticas
18 int contadordeTrapecios = 1; // Contador global para el nmero de trapecios
19 string n_archivo = "TrapecioParalelo.dat"; // Nombre del archivo de salida
20 int decimales = 6; // Nmero de decimales a usar
21
22 // Definición de la clase Trapecio
23 class Trapecio {
24 private:
25     double altura; // Altura del trapecio
26     double basemayor; // Base mayor del trapecio
27     double basemenor; // Base menor del trapecio
28 public:
29     // Constructor que inicializa las dimensiones del trapecio
30     Trapecio(double _altura, double _basemayor, double _basemenor): altura(_altura),
31         basemayor(_basemayor), basemenor(_basemenor) {}
32 }
```



```
32 // Mtodo que calcula el rea del trapecio
33 double calculararea() { return ((basemayor + basemenor) * altura) / 2; // Frmula para el
    rea del trapecio
34 }
35 };
36
37 // Funcin que define la ecuacin a integrar
38 double y(double x) {
39     return 2 * pow(x, 2) + 1; // Retorna el valor de la funcin en x
40 }
41
42 // Funcin para generar un archivo con los tiempos y reas calculadas
43 void generararchivo(vector<double> tiempos, vector<double> areas) {
44     ofstream myfile(n_archivo); // Abre el archivo para escribir
45     int sizet = tiempos.size(); // Obtiene el tamao del vector de tiempos
46     for (int i = 0; i < sizet; i++) { // Itera sobre los elementos del vector
47         cout << "Con " << i + 1 << " trapecios, el area calculada es: " << areas[i] << endl;
            // Imprime el rea calculada
48         if (i != sizet - 1) {
49             myfile << i + 1 << "\t" << tiempos[i] << endl; // Escribe el nmero de trapecios y
                el tiempo en el archivo
50         } else {
51             myfile << i + 1 << "\t" << tiempos[i]; // ltima lnea sin salto de lnea
52         }
53     }
54     myfile.close(); // Cierra el archivo
55     cout << endl;
56 }
57
58 // Funcin para calcular el rea usando n trapecios
59 double calculartrapecio_n(double contadordetrapecios, double altura, double &temp, double
    sum_areas, double limiteinferior, double limitesuperior) {
60     for (int i = 0; i < contadordetrapecios; i++) { // Itera sobre el nmero de trapecios
61         unique_ptr<Trapecio> trapecio(new Trapecio(altura, y(temp + altura), y(temp))); //
            Crea un nuevo objeto Trapecio
62         double area_actual = trapecio->calculararea(); // Calcula el rea del trapecio actual
63         sum_areas = sum_areas + area_actual; // Suma el rea actual al total
64         temp = temp + altura; // Incrementa el valor de temp
65     }
66     return sum_areas; // Retorna la suma de las reas
67 }
68
69 // Funcin para agregar datos a los vectores de tiempos y reas
70 void pusheardatos(vector<double>& tiempos, double tiempo, vector<double>& areas, double
    sum_areas) {
71     tiempos.push_back(tiempo); // Agrega el tiempo al vector de tiempos
72     areas.push_back(sum_areas); // Agrega el rea al vector de reas
73 }
74
75 // Funcin para calcular el rea total bajo la curva
76 void areatotal(double limiteinferior, double limitesuperior) {
77     thread threads[n_threads]; // Crea un array de hilos
78     vector<double> areas; // Vector para almacenar las reas calculadas
79     vector<double> tiempos; // Vector para almacenar los tiempos de clculo
80     double tolerancia = pow(10, -decimales); // Tolerancia para la comparacin de reas
81 }
```

```
82 while (true) { // Bucle infinito hasta que se cumpla la condicin de parada
83     double sum_areas = 0.0; // Inicializa la suma de reas
84     double intervalo_por_hilo = (limitesuperior - limiteinferior) / n_threads; // Calcula
85     // el intervalo que cada hilo procesar
86     double sum_areas_hilos[n_threads] = {0.0}; // Array para almacenar las reas
87     // calculadas por cada hilo
88
89     // Iniciar cronmetro antes de que los hilos comiencen
90     auto start = chrono::steady_clock::now();
91
92     // Iniciar hilos
93     for (int i = 0; i < n_threads; ++i) {
94         double inicio_hilo = limiteinferior + i * intervalo_por_hilo; // Calcula el inicio
95         // del intervalo del hilo
96         double fin_hilo = inicio_hilo + intervalo_por_hilo; // Calcula el fin del
97         // intervalo del hilo
98         if (i == n_threads - 1) fin_hilo = limitesuperior; // Ajustar el ltimo hilo
99
100        threads[i] = thread([&sum_areas_hilos, i, inicio_hilo, fin_hilo]() { // Lambda
101            // para ejecutar en cada hilo
102            double sum_areas_local = 0.0; // Inicializa la suma de reas local
103            int contadordetrapecios_local = contadordetrapecios; // Copia local del
104            // contador de trapecios
105            double altura_local = (fin_hilo - inicio_hilo) / contadordetrapecios_local; //
106            // Calcula la altura local de cada trapecio
107            for (int j = 0; j < contadordetrapecios_local; ++j) { // Itera sobre el nmero
108            // de trapecios
109                double temp_local = inicio_hilo + j * altura_local; // Calcula el valor
110                // temporal local
111                double base_mayor = y(temp_local + altura_local); // Calcula la base mayor
112                double base_menor = y(temp_local); // Calcula la base menor
113                double area_actual = (base_mayor + base_menor) * altura_local / 2.0; //
114                // Calcula el rea del trapecio actual
115                sum_areas_local += area_actual; // Suma el rea local al total local
116            }
117            // Actualizar la suma total de reas con mutex
118            mtx.lock();
119            sum_areas_hilos[i] = sum_areas_local; // Guarda el resultado local en el array
120            // compartido
121            mtx.unlock();
122        });
123    }
124
125    // Esperar a que todos los hilos terminen
126    for (int i = 0; i < n_threads; ++i) {
127        threads[i].join();
128    }
129
130    // Detener el cronmetro despues de que todos los hilos hayan terminado
131    auto end = chrono::steady_clock::now();
132
133    // Calcular la suma total de las reas de los trapecios
134    for (int i = 0; i < n_threads; ++i) {
135        sum_areas += sum_areas_hilos[i]; // Suma las reas calculadas por cada hilo
136    }
```

```
127     auto duracion_ns = duration_cast<nanoseconds>(end - start).count(); // Calcula la
128         duracin en nanosegundos
129     pusheardatos(tiempos, duracion_ns, areas, sum_areas); // Agrega los datos a los
130         vectores de tiempos y reas
131
132     if (areas.size() > 1 && abs(areas[contadordetrapecios - 1] - areas[contadordetrapecios
133         - 2]) < tolerancia) { // Verifica la condicin de tolerancia
134         generararchivo(tiempos, areas); // Genera el archivo con los datos
135         cout << "El rea total calculada entre los intervalos " << limiteinferior << " a "
136             << limitesuperior << " es aproximadamente: " << areas[contadordetrapecios -
137             1] << " unidades cuadradas" << endl; // Imprime el resultado final
138         cout << "Trapecios mximos usados: " << contadordetrapecios << " trapecios" <<
139             endl; // Imprime el nmero de trapecios usados
140         cout << "Decimales trabajados: " << 6 << endl; // Imprime la precisin en decimales
141         cout << "Tolerancia usada: " << tolerancia << endl; // Imprime la tolerancia usada
142         cout << endl;
143         cout << "Archivo " << n_archivo << " creado" << endl; // Informa que el archivo
144             fue creado
145         break; // Rompe el bucle
146     }
147
148     contadordetrapecios++; // Incrementa el contador de trapecios
149 }
150
151 // Funcin para mostrar un saludo segn la hora del da
152 void saludarsegunhora() {
153     time_t ahora = time(0); // Obtiene el tiempo actual
154     tm *horaLocal = localtime(&ahora); // Convierte el tiempo a la hora local
155     int hora = horaLocal->tm_hour; // Obtiene la hora actual
156     if (hora >= 6 && hora < 12) {
157         cout << "Buenos das, al sistema para hallar la integral definida aproximada con el
158             mtodo del trapecio" << endl; // Saludo de buenos das
159     } else if (hora >= 12 && hora < 18) {
160         cout << "Buenas tardes, al sistema para hallar la integral definida aproximada con el
161             mtodo del trapecio" << endl; // Saludo de buenas tardes
162     } else {
163         cout << "Buenas noches, al sistema para hallar la integral definida aproximada con el
164             mtodo del trapecio" << endl; // Saludo de buenas noches
165     }
166 }
167
168 // Manipulador de flujo para establecer la precisin a 6 decimales
169 ostream& precision_six(ostream& os) {
170     os << fixed << setprecision(6); // Establece la precisin a 6 decimales
171     return os; // Retorna el flujo
172 }
173
174 // Funcin principal
175 int main() {
176     cout << precision_six; // Aplica el manipulador de flujo para precisin
177     saludarsegunhora(); // Llama a la funcin para mostrar el saludo
178     while(true) { // Bucle infinito hasta que se cumpla la condicin de salida
179         double limitesuperior; // Variable para el lmite superior
180         double limiteinferior; // Variable para el lmite inferior
181         cout << "Digite el lmite inferior: ";
182         cin >> limiteinferior; // Lee el lmite inferior
```

```
173     cout << "Digite el lmite superior: ";
174     cin >> limitesuperior; // Lee el lmite superior
175     cout << endl;
176     if(limitesuperior == limiteinferior) { // Verifica si los lmites son iguales
177         cout << "El rea total calculada entre los intervalos " << limiteinferior << " a "
178             << limitesuperior << " es aproximadamente: " << 0 << " unidades cuadradas" <<
179             endl; // Imprime que el rea es cero
180         break; // Rompe el bucle
181     }
182     if(limitesuperior > limiteinferior) { // Verifica si el lmite superior es mayor que
183         el inferior
184         cout << "Calculando...." << endl; // Imprime mensaje de clculo
185         areatotal(limiteinferior, limitesuperior); // Llama a la funcin para calcular el
186         rea total
187         cout << endl;
188         break; // Rompe el bucle
189     } else {
190         cout << "Valores incorrectos o no lgicos." << endl; // Imprime mensaje de error
191         cout << "El lmite superior debe ser mayor que el lmite inferior." << endl; //
192         Indica el error en los valores
193         cout << endl;
194         cout << "Ingrese de nuevo los parmetros correctos:" << endl; // Pide ingresar
195         nuevamente los valores
196     }
197 }
198 return 0; // Retorna 0 indicando que el programa termin correctamente
199 }
```

9.2.2. Ejecución del algoritmo paralela:

Para la ejecución del algoritmo, en nuestro entorno de desarrollo preferido, o en consola, podemos ejecutar el programa de la siguiente manera:

Listing 8: Ejecutar Trapeciolineal.cpp

```
1 C:\lp3-24a\parcial\exercises
2 g++ -o Trapecioparalelo.exe Trapecioparalelo.cpp
3 Trapecioparalelo.exe
```

Esto nos dara el resultado final, conjuntamente creandonos un archivo TrapecioParalelo.dat, este archivo lo podemos encontrar en el output de nuestro entorno de desarrollo, o si estamos desarrollando online, en la ventana vecina.

Listing 9: Salida del programa:

```
1 Ingrese el lmite inferior: 0
2 Ingrese el lmite superior: 10
```

Esto nos dara el resultado final, conjuntamente creandonos un archivo TrapecioParalelo.dat, este archivo lo podemos encontrar en el output de nuestro entorno de desarrollo, o si estamos desarrollando online, en la ventana vecina.

Listing 10: Salida del programa:

```
1  Calculando.....
2  .....
3  .....
4  .....
5  Con 395 trapecios, el area calculada es: 676.66880
6  Con 396 trapecios, el area calculada es: 676.66879
7  Con 397 trapecios, el area calculada es: 676.66878
8  Con 398 trapecios, el area calculada es: 676.66877
9  Con 399 trapecios, el area calculada es: 676.66876
10 Con 400 trapecios, el area calculada es: 676.66875
11 Con 401 trapecios, el area calculada es: 676.66874
12 Con 402 trapecios, el area calculada es: 676.66873
13 Con 403 trapecios, el area calculada es: 676.66872
14 Con 404 trapecios, el area calculada es: 676.66871
15 Con 405 trapecios, el area calculada es: 676.66870
16 Con 406 trapecios, el area calculada es: 676.66869
17
18 El area total calculada entre los intervalos 0.00000 a 10.00000 es: 676.66869 unidades
   cuadradas
19 Trapecios maximos usados: 406 trapecios
20
21 << El programa ha finalizado: codigo de salida: 0 >>
22 << Presione enter para cerrar esta ventana >>
```

10. Gnuplot

Es una herramienta de trazado de gráficos que permite generar gráficos bidimensionales y tridimensionales de funciones matemáticas y conjuntos de datos. Utilizando comandos simples y scripts, Gnuplot puede producir una amplia variedad de gráficos, incluyendo gráficos de dispersión, gráficos de líneas, gráficos de barras, histogramas, superficies y más.

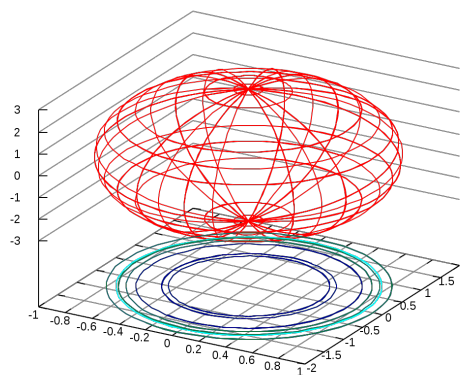


Imagen con propiedad de ©Wikipedia

10.1. Comandos comunes en gnuplot:

1.- plot: Este comando se utiliza para trazar gráficos. Por ejemplo, `plot sin(x)` traza el gráfico de la función seno.

2.- set: este comando se utiliza para configurar diversas opciones del gráfico, como el rango de los ejes, los títulos de los ejes, los estilos de línea, los colores, etc. Por ejemplo, `set xlabel Eje X` establece la etiqueta del eje X.

3.- unset: este comando se utiliza para desactivar opciones previamente configuradas. Por ejemplo, `unset key` desactiva la leyenda del gráfico.

4.- splot: similar al comando `plot`, pero se utiliza para trazar gráficos tridimensionales.

5.- replot: este comando se utiliza para volver a trazar el gráfico actual. Es útil cuando se realizan cambios en el gráfico y se desea actualizar la visualización.

6.- pause: este comando pausa la ejecución del script durante un número específico de segundos. Es útil para controlar la velocidad de la visualización.

7.- help: este comando muestra la documentación de Gnuplot. Por ejemplo, `help plot` muestra información sobre el comando `plot`.

10.2. Uso de gnuplot en nuestro laboratorio:

En este laboratorio, utilizaremos Gnuplot para generar gráficos y visualizar datos de experimentos y simulaciones. Gnuplot nos permite representar de forma clara y precisa los resultados obtenidos, lo que facilita el análisis y la interpretación de los datos en los tiempos de los algoritmos de ordenamiento.

Para utilizar Gnuplot en nuestro laboratorio, seguimos los siguientes pasos:

- 1.- Instalar gnuplot en la computadora local.
- 2.- Preparación de los datos `.dat`.
- 3.- Ejecutar los comandos de gnuplot para la elaboración de gráficos 2D.
- 4.- Analizar e interpretar.

10.2.1. Instalar gnuplot en la computadora local:

Para Windows, podemos descargar el instalador ejecutable de la página oficial de gnuplot: <https://sourceforge.net/projects/gnuplot/>. Una vez descargado, ejecutamos el instalador y seguimos las instrucciones en pantalla para completar la instalación.

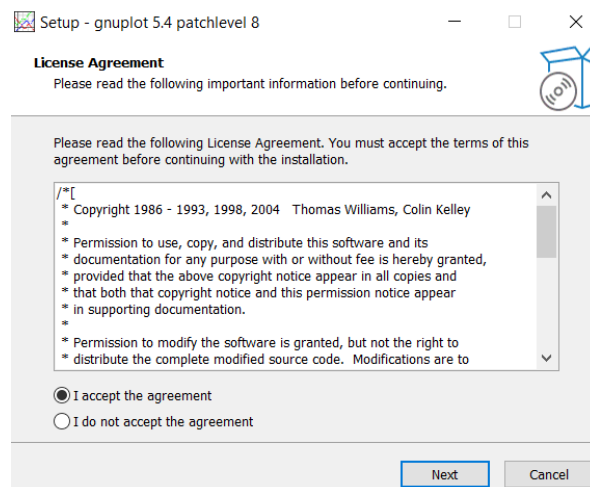


Imagen con propiedad de ©gnuplot

10.2.2. Preparación de los datos .dat.

Para preparar los datos en formato .dat que necesitamos para realizar las gráficas con Gnuplot, debemos ejecutar nuestros programas en C++ de la siguiente manera:

Trapeciolineal:

Listing 11: Generar InsertionSort.dat

```
1 C:\lp3-24a\parcial\exercises
2 g++ -o Trapeciolineal.exe Trapeciolineal.cpp
3 Trapeciolineal.exe
```

Trapecioparalelo:

Listing 12: Generar QuickSort.dat

```
1 C:\lp3-24a\parcial\exercises
2 g++ -o Trapecioparalelo.exe Trapecioparalelo.cpp
3 Trapecioparalelo.exe
```

Una vez esperado los tiempos de ejecución, cada uno de los programas nos arrojan los siguientes resultados:

10.2.3. Ejecutar los comandos de gnuplot para la elaboracion de graficos 2D:

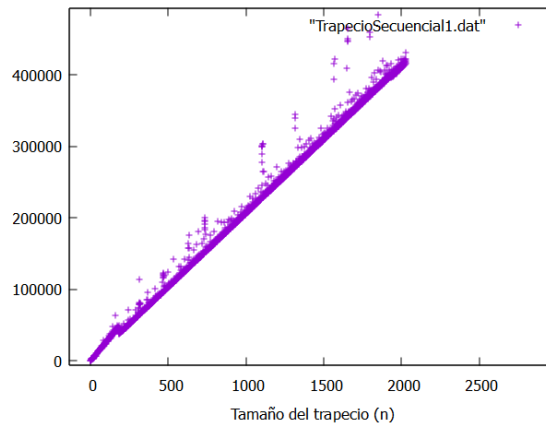
Una vez obtenido los .dat de cada ejecución de programa correspondiente, podemos utilizar el gnuplot para graficar e interpretar los gráficos correspondientes:

Listing 13: Cargar los datos a gnu plot

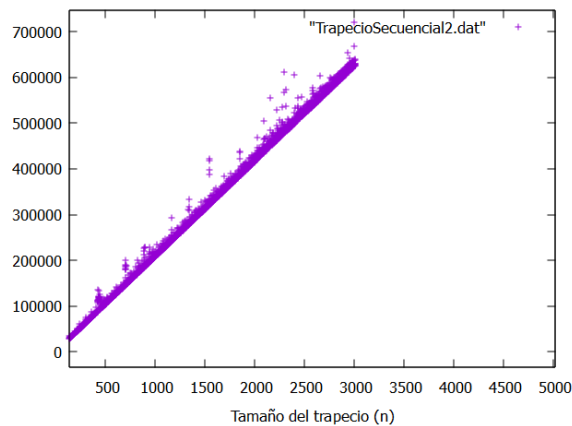
```
1 Terminal type is now 'qt'
2 gnuplot> cd 'C:\lp3-24a\parcial\dat'
3 gnuplot> set title "Análisis y gnuplot> comportamiento de paradigmas de programacion"
4 gnuplot> set xlabel "Tamaño del trapecio (n)"
5 gnuplot> set ylabel "Tiempo de calculo (nanosegundos)"
6 gnuplot> plot "TrapecioSecuencial.dat" with lines
7 gnuplot> replot "TrapecioParalelo.dat" with lines
```

Salida del programa gnuplot:

Programacion secuencial: mostrando diferentes casos



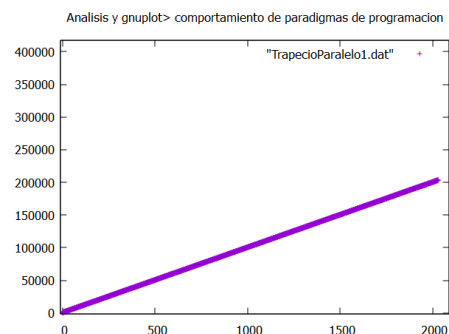
programación secuencial, límite inferior 0, límite superior=50, precisión 5, integral $2(x*x)+1$



programación secuencial, límite inferior 3, límite superior=50, precisión 4, integral $x*x*x$

Podemos observar que tanto en los dos casos de la programación secuencial, se muestra una curva positiva incremental, esto significa que mientras mas trapecios se usen para encontrar la integral aproximada, demandará mas tiempo en poder encontrar el resultado esperado. Esto a la larga produce cierto nivel de ineficiencia.

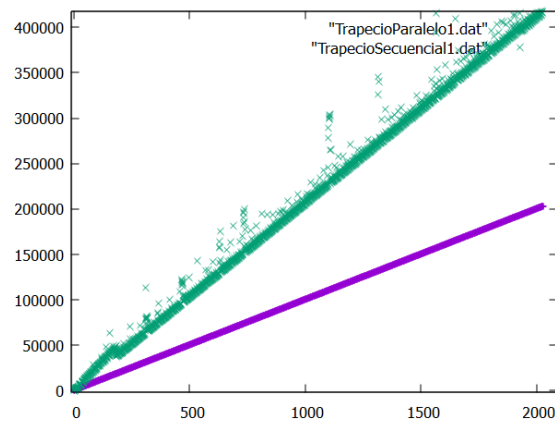
Programación paralela: mostrando diferentes casos



programación secuencial, límite inferior 0, límite superior=50, precisión 5, integral $2(x*x)+1$

10.2.4. Analizar e interpretar los datos obtenidos:

Programación paralela: mostrando diferentes casos



COMPARACION PARALELA VS SECUENCIAL

En este gráfico se observa claramente cómo la programación paralela supera a la secuencial en términos de eficiencia y velocidad de procesamiento. La línea correspondiente a la programación paralela muestra una tendencia descendente, lo que indica que a medida que aumenta la cantidad de recursos computacionales (como el número de núcleos de CPU o hilos), el tiempo de ejecución disminuye significativamente. Por otro lado, la línea que representa la programación secuencial tiende a mantenerse más constante o incluso a aumentar ligeramente a medida que se incrementan los recursos computacionales, lo que sugiere que la mejora en el tiempo de ejecución es limitada en comparación con la programación paralela.

11. Estructura del parcial:

```
1 |--- parcial
2 |   |--- dat [DIRECTORY]
3 |       |--- TrapecioSecuencial1.dat
4 |       |--- TrapecioSecuencial2.dat
5 |   |--- report [DIRECTORY]
6 |       |--- partial.pdf
7 |       |--- partial.zip
8 |   |--- exercises [DIRECTORY]
9 |       |--- Trapeciolineal.cpp
10 |       |--- Trapecioparalelo.cpp
11 3 directories, 7 files
```

12. Referencias:

- https://www.onlinegdb.com/online_c++_compiler
- <https://sourceforge.net/projects/gnuplot/>
- <https://github.com/>
- <https://pseint.sourceforge.net/>

- <https://www.wikipedia.org/>

13. Calificación del docente:

Tabla 1: Rúbrica para contenido del Informe y evidencias

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	El repositorio se pudo clonar y se evidencia la estructura adecuada para revisar los entregables. (Se descontará puntos por error o observación)	4	X		
2. Commits	Hay porciones de código fuente asociado a los commits planificados con explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X		
3. Ejecución	Se incluyen comandos para ejecuciones y pruebas del código fuente explicadas gradualmente que permitirían replicar el proyecto. (Se descontará puntos por cada omisión)	4	X		
4. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X		
5. Ortografía	El documento no muestra errores ortográficos. (Se descontará puntos por error encontrado)	2	X		
6. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente con explicaciones puntuales pero precisas, agregando diagramas generados a partir del código fuente y refleja un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X		
Total		20			