

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_trapezio
```

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_trapezio  
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
	Trapezio	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.

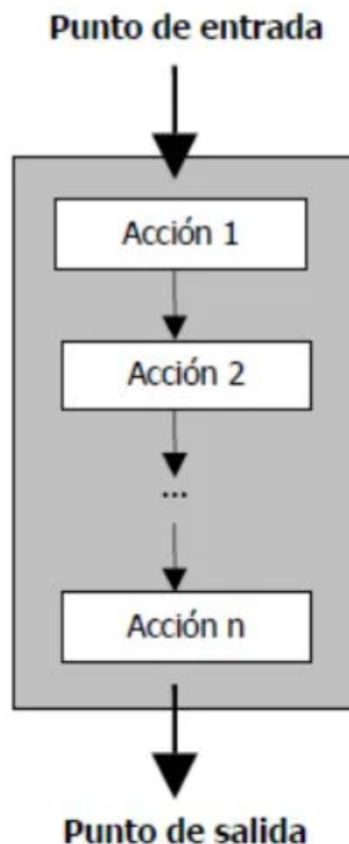


Imagen con propiedad de ©Issuu

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.

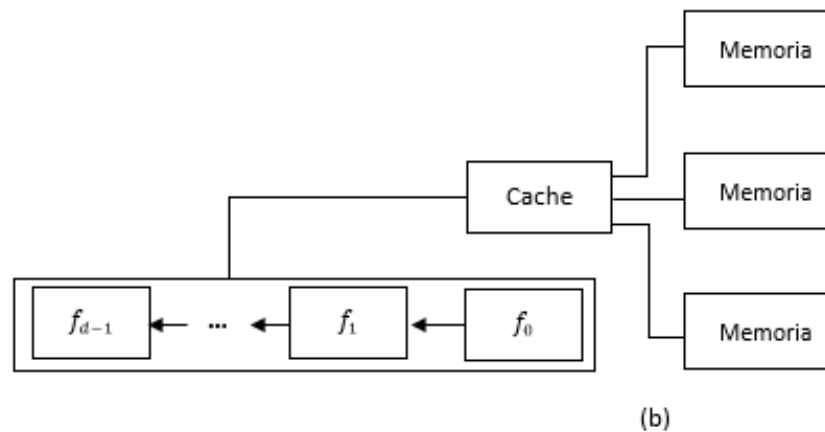


Imagen con propiedad de ©procesamientoparalelo.blogspot

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.

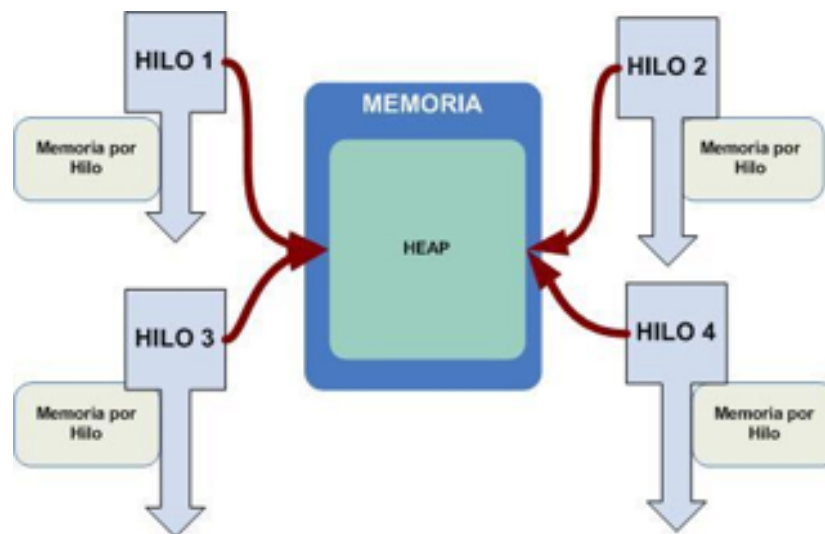


Imagen con propiedad de ©fERESTREPOCA

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.

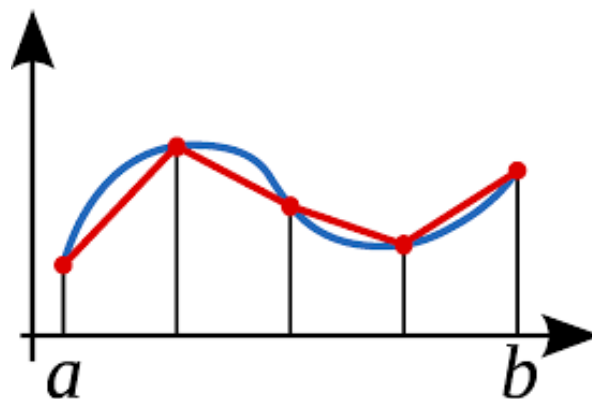


Imagen con propiedad de ©Antonio Cedillo Hernandez

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:

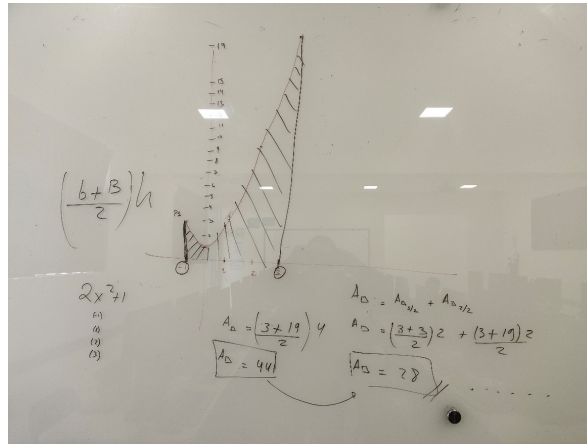


Imagen con propiedad de ©Richart Escobedo

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 $y(\text{double } x)$: 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:

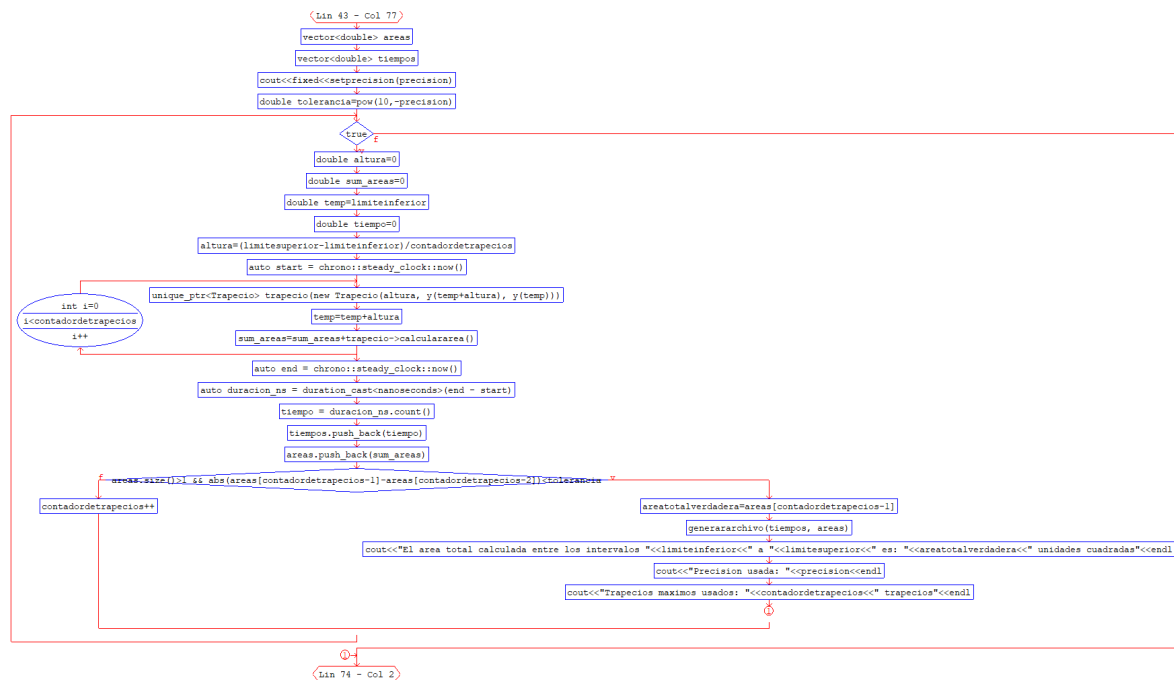


Imagen con propiedad de ©Rodrigo Infanzon, ©Zinjal

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 final: 058a45707c82def339e59c0da9c1e9f3f498d796

En este ultimo commit, se presenta la explicación del programa por medio de comentarios, esto ayu-

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

9.1.3. Código fuente: programación secuencial, método del trapecio

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

```
1 #include <iostream>    // Biblioteca estandar de entrada y salida en C++, para entrada/salida
   basica.
2 #include <memory>      // Para la gestion de memoria dinamica y la utilizacion de punteros
   inteligentes.
3 #include <iomanip>     // Proporciona facilidades para la manipulacion de formatos de
   entrada/salida.
4 #include <cmath>      // Contiene funciones matematicas comunes, como sqrt(), sin(), cos(), etc.
5 #include <vector>     // Ofrece una clase de contenedor de secuencia dinamica, similar a los
   arrays, pero de longitud dinamica.
6 #include <fstream>    // Proporciona las clases necesarias para realizar operaciones de
   entrada/salida en archivos.
7 #include <ctime>      // Utilizado para trabajar con funciones y estructuras relacionadas con el
   tiempo y la fecha.
8 #include <chrono>     // Biblioteca para medir el tiempo y realizar calculos temporales de alta
   precision.
9
10
11 using namespace std;
12 using namespace chrono;
13
14 // Inicializamos nuestra variable global contador de trapecios, esta nos sirve para poder
   saber cuantos trapecios hemos necesitado para realizar el calculo
15 int contadordetrapecios=1;
16
17 // Inicializamos nuestra variable global areatotalverdadera, esta nos sirve para retornar el
   area verdadera aproximada entre la funcion y los limites inferior y superior
18 double areatotalverdadera=0;
19
20 // Definimos nuestra clase trapecio, en P00 es muy importante, ya que podemos tratar a este
   trapecio como objeto, ademas es facil de poder entenderlo
21 class Trapecio {
22 // Definimos los atributos privados para el trapecio : altura, base menor y base mayor
23 private:
24     double altura;
25     double basemayor;
26     double basemenor;
27 // Creamos su constructor publico, Trapecio, ademas agregamos una funcion llamada
   calculararea, esta tiene el proposito de calcular el area de un trapecio
28 public:
29     Trapecio(double _altura, double _basemayor, double _basemenor): altura(_altura),
        basemayor(_basemayor), basemenor(_basemenor) {}
30
31 // calculararea recibira basemayor, basemenor y altura, y hara el calculo matematico
   respectivo
32     double calculararea() {return ((basemayor+basemenor)*altura)/2;}
33 };
34
35 // Funcion f(x)
36 double y(double x) {return 2*pow(x,2)+1;}
```

```
37
38 // Funcion generar archivo: posee parametros tales como: tiempos, areas
39 void generararchivo(vector<double> tiempos, vector<double> areas){
40     ofstream myfile("TrapezioSecuencial.dat"); // usando <fstream> creamos un archivo.dat
41     int sizet= tiempos.size(); // del vector tiempos, sacamos su longitud usando la funcion
42     .size()
43     for(int i=0;i<sizet;i++) { // inicializamos un for que va desde 0 hasta la longitud del
44         vector tiempos, esto hace recorrer cada tiempo ya calculado
45         cout<<"Con "<<i+1<<" trapecios, el area calculada es: "<<areas[i]<<endl; //
46             imprimimos el numero de trapezio y su area calculada
47         if(i!=sizet-1) { // si i es diferente de la longitud del vector entonces seteamos el
48             .dat
49             myfile<<i+1<<"\t"<<tiempos[i]<<endl;
50         } else {
51             myfile<<i+1<<"\t"<<tiempos[i]; // caso contrario seteamos el ultimo elemento sin
52             aadir un salto de linea
53         }
54     }
55     myfile.close(); // cerramos el archivo
56     cout<<endl; // agregamos un salto de linea
57 }
58
59 // Funcion areatotal: posee parametros tales como limiteinferior, limitesuperior, y el nivel
60 de presicion del calculo general
61 void areatotal(double limiteinferior, double limitesuperior, int precision) {
62     vector<double> areas; // inicializamos un vector areas, esta nos servira para almacenar
63     todas las areas calculadas con n= 1,2,3,4,5,6... etc. trapecios
64     vector<double> tiempos; // inicializamos nuestro vector tiempos, esta nos servira
65     cout<<fixed<<setprecision(precision); // esto indica cuantos digitos se deben imprimir
66     despues del punto decimal.
67     double tolerancia=pow(10,-precision); // calcular la tolerancia a partir de la precision
68     establecida
69     while(true) {
70         double altura=0; // inicializamos nuestra altura en 0
71         double sum_areas=0; // inicializamos nuestra suma de areas, esta nos sirve para
72             almacenar la suma total de areas de cada trapezio pequeno calculado
73         double temp=limiteinferior; // inicializamos temp que sera igualado a nuestro limite
74             inferior
75         double tiempo=0; // inicializamos tiempo, esto nos sirve para calcular el tiempo que
76             despues sera almacenado en nuestro vector tiempos
77         altura=(limitesuperior-limiteinferior)/contadordetrapecios; // la altura sera
78             calculada con la resta de los limites dividido entre el actual numero de
79             trapecios, esto nos ayuda a saber la altura total de cada pequeno trapezio
80         auto start = chrono::steady_clock::now(); // inicializamos la toma de tiempo (start)
81             osea inicio
82         for(int i=0;i<contadordetrapecios;i++) { // inicializamos nuestro for, que va desde
83             i=0 hasta el valor actual del contador de trapecios
84             unique_ptr<Trapezio> trapezio(new Trapezio(altura, y(temp+altura), y(temp))); //
85                 creamos un puntero inteligente trapezio, esto nos ayuda a que despues este
86                 sea utilizado, libere la memoria automaticamente
87             temp=temp+altura; // se hace un nuevo calculo del temporal para usarlo en el
88                 siguiente trapezio
89             sum_areas=sum_areas+trapezio->calculararea(); // usamos la funcion calculararea,
90                 para hallar el area del trapezio actual de la iteracion y lo sumamos al
91                 sum_areas
92         }
93     }
```

```
72     auto end = chrono::steady_clock::now(); // acabamos la toma de tiempo (end) osea final
73     auto duracion_ns = duration_cast<nanoseconds>(end - start); // restamos el tiempo
        final - el tiempo inicial (en nanosegundos)
74     tiempo = duracion_ns.count(); // count para devolver el numero de periodos
75     tiempos.push_back(tiempo); // agregamos el tiempo calculado al vector de tiempos
76     areas.push_back(sum_areas); // agregamos el area calculada al vector de areas
77     if (areas.size()>1 &&
        abs(areas[contadordetrapecios-1]-areas[contadordetrapecios-2])<tolerancia) { //
        si la longitud del vector areas es mayor que 1 y el valor absoluto de la resta
        entre el actual calculo de area menos el anterior es menor que la tolerancia
        entonces se termina el calculo
78     areatotalverdadera=areas[contadordetrapecios-1]; // seteamos el calculo actual a
        areatotalverdadera
79     generararchivo(tiempos, areas); // llamamos a la funcion generar archivo
80     // Imprimimos los valores para la consola
81     cout<<"El area total calculada entre los intervalos "<<limiteinferior<<" a
        "<<limitesuperior<<" es: "<<areatotalverdadera<<" unidades cuadradas"<<endl;
        // area total calculada
82     cout<<"Precision usada: "<<precision<<endl; // precision usada
83     cout<<"Trapecios maximos usados: "<<contadordetrapecios<<" trapecios"<<endl; //
        trapecios usados
84     break; // rompemos el bucle
85     } else {contadordetrapecios++;} // en caso el calculo sea mayor que la tolerancia, el
        contador se incrementa en 1 pasando denuevo por el bucle
86 }
87 }
88
89 // Funcion para saludar segun la hora ejecutada por el usuario
90 void saludarsegunhora() {
91     time_t ahora=time(0); // usamos la libreria <ctime>
92     tm *horaLocal=localtime(&ahora); // convertimos el tiempo actual en una estructura tm que
        representa la hora local, localtime() convierte la representacin de tiempo en el
        sistema local
93     int hora=horaLocal->tm_hour; // extraer la hora del objeto tm obtenido anteriormente
94     // Verificamos en qu rango horario est la hora actual y mostrar un mensaje correspondiente
95     if (hora>=6 && hora<12) {
96         cout<<"Buenos das, al sistema para hallar la integral definida aproximada con el
            metodo del trapecio"<<endl;
97     } else if (hora>=12 && hora<18) {
98         cout<<"Buenas tardes, al sistema para hallar la integral definida aproximada con el
            metodo del trapecio"<<endl;
99     } else { cout<<"Buenas noches, al sistema para hallar la integral definida aproximada con
        el metodo del trapecio"<<endl;}
100 }
101
102 int main() {
103     saludarsegunhora(); // saludamos segun la hora en que se ejecuta el programa
104     while (true) {
105         double limitesuperior; // inicializamos limitesuperior
106         double limiteinferior; // inicializamos limiteinferior
107         double precision; // inicializamos presicion
108         cout<<"Digite el limite inferior: ";
109         cin>>limiteinferior; // leemos limiteinferior
110         cout<<"Digite el limite superior: ";
111         cin>>limitesuperior; // leemos limitesuperior
112         cout<<endl;
```

```
113 // Hacemos una comparacion para determinar que los datos esten correctamente
    establecidos
114 if (limitesuperior>limiteinferior) {
115     while(true) {
116         cout<<"Digite el nivel de precision"<<endl;
117         cout<<"Entre mas alto sea el nivel de precision, se incrementara el numero de
            trapecios a calcular y"<<endl;
118         cout<<"por ende el tiempo"<<endl;
119         cout<<"Nivel de precision: ";
120         cin>>precision; // leemos la presicion
121         // Determinamos que la presicion este correctamente establecida
122         if (precision>0) {
123             cout<<endl;
124             cout<<"Calculando...."<<endl;
125             areatotal(limiteinferior, limitesuperior, precision);
126             cout<<endl;
127             break; // caso en que la presicion este correctamente establecida, se
                rompe el bucle y se inician los calculos
128         } else {
129             cout<<"Precision no valida, por favor ingrese un valor positivo."<<endl;
                // caso no este correcta se debe de volver a establecer o leer
130         }
131     }
132 } else {
133     cout<<"Valores incorrectos o no lgicos."<<endl;
134     cout<<"El limite superior debe ser mayor que el limite inferior."<<endl;
135     cout<<endl;
136     cout<<"Ingrese denuevo los parametros correctos:"<<endl; // caso los limites no
        esten correctamente establecidos, se deben de volver a leer
137 }
138 break; // caso todo este correcto, se inicia los calculos y se termina el programa
139 }
140 return 0;
141 }
```

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
4
```

```
5  Digite el nivel de precision
6  Entre mas alto sea el nivel de precision, se incrementara el numero de trapecios a
   calcular y
7  por ende el tiempo
8  Nivel de precision: 5
```

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 Precision usada: 5
20 Trapecios maximos usados: 406 trapecios
21
22 << El programa ha finalizado: codigo de salida: 0 >>
23 << 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 código, 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 para mejorar el rendimiento. Se divide el intervalo de integración en múltiples trapecios y se distribuye el cálculo entre varios hilos, utilizando el método del trapecio para cada subintervalo. Esto se logra mediante la función `areatotalparalela()`, que utiliza la biblioteca `pthread` para crear y administrar los hilos. La variable `totalArea` se utiliza de manera atómica para garantizar la consistencia al acumular las áreas calculadas por cada hilo.

Para evaluar el rendimiento del cálculo en paralelo, se implementa la función `medirTiempos()`, que mide el tiempo de ejecución para diferentes cantidades de trapecios. Los resultados se escriben en un archivo de texto llamado `TrapecioParalelo.dat`, lo que permite analizar el comportamiento del algoritmo en función del número de trapecios utilizados en la aproximación.

Finalmente, en la función `main()`, se solicitan al usuario los límites inferior y superior de la función a integrar. Se establece un número máximo de trapecios para la aproximación y se llama a `medirTiem-`

pos() para evaluar el rendimiento del cálculo en paralelo. Además, se muestra el área total calculada utilizando el número máximo de trapecios definido.

El código muestra una implementación eficiente y escalable del método del trapecio utilizando paralelismo, lo que permite una aproximación precisa del área bajo una curva de función en un intervalo dado. Sin embargo, podrían realizarse mejoras adicionales, como comparaciones con versiones secuenciales del algoritmo o técnicas de optimización adicionales para mejorar aún más el rendimiento.

9.2.1. Commits importantes del código:

Ultimo commit: **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.

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

```
1 #include <iostream>
2 #include <iomanip>
3 #include <fstream>
4 #include <chrono>
5 #include <cmath>
6 #include <vector>
7 #include <atomic>
8 #include <thread>
9 #include <memory>
10 using namespace std;
11 using namespace chrono;
12
13 // Definición de la función y(x)
14 double y(double x) {
15     return 2 * pow(x, 2) + 1;
16 }
17
18 // Definición de la clase Trapecio
19 class Trapecio {
20 private:
21     double altura;
22     double basemayor;
23     double basemenor;
24 public:
25     Trapecio(double _altura, double _basemayor, double _basemenor): altura(_altura),
        basemayor(_basemayor), basemenor(_basemenor) {}
26
27     double calculararea() {return ((basemayor + basemenor) * altura) / 2;}
28 };
29
```



```
30 // Funcin que calcula el rea total en paralelo utilizando el mtodo del trapecio
31 double areatotalparalela(int trapezoids, double lowerLimit, double upperLimit){
32     double p = (upperLimit - lowerLimit) / trapezoids;
33     atomic<double> totalArea(0.0);
34     atomic<int> next(0);
35     int numThreads = thread::hardware_concurrency();
36     unique_ptr<thread[]> threads(new thread[numThreads]);
37
38     auto worker = [&]() {
39         while(true) {
40             int i = next.fetch_add(1);
41             if(i >= trapezoids){
42                 break;
43             }
44             double x = lowerLimit + i * p;
45             unique_ptr<Trapezio> trapezoid(new Trapecio(p, y(x + p), y(x)));
46             double area = trapezoid->calculararea();
47             double currentTotal = totalArea.load();
48             while(!totalArea.compare_exchange_weak(currentTotal, currentTotal + area)) {}
49         }
50     };
51
52     for (int i = 0; i < numThreads; i++){
53         threads[i] = thread(worker);
54     }
55
56     for(int i = 0; i < numThreads; i++){
57         threads[i].join();
58     }
59
60     return totalArea;
61 }
62
63
64 // Funcin para medir tiempos de ejecucin para diferentes cantidades de trapecios
65 void medirTiempos(double limiteinferior, double limitesuperior) {
66     // Definir la cantidad mxima de trapecios a probar
67     int maxTrapecios = 100000;
68
69     // Abriendo un archivo para escribir los resultados
70     ofstream myfile;
71     myfile.open("TrapezioParalelo.dat");
72
73     // Iterar sobre diferentes cantidades de trapecios y medir el tiempo
74     for (int trapecios = 1; trapecios <= maxTrapecios; trapecios++) {
75         auto start_time = high_resolution_clock::now();
76
77         double area = areatotalparalela(trapecios, limiteinferior, limitesuperior);
78
79         auto end_time = high_resolution_clock::now();
80         auto duration = duration_cast<nanoseconds>(end_time - start_time);
81
82         // Escribir en el archivo
83         myfile << trapecios << "\t" << duration.count() << endl;
84     }
85 }
```

```
86 // Cerrar el archivo
87 myfile.close();
88 }
89
90
91 int main() {
92     double limiteinferior, limitesuperior;
93     cout << "Ingrese el lmite inferior: ";
94     cin >> limiteinferior;
95     cout << "Ingrese el lmite superior: ";
96     cin >> limitesuperior;
97
98     int maxTrapecios = 1000000; // Definir la cantidad mxima de trapecios a probar
99     medirTiempos(limiteinferior, limitesuperior);
100
101     cout << "Area total calculada: " << areatotalparalela(maxTrapecios, limiteinferior,
102         limitesuperior) << endl;
103
104     return 0;
105 }
```

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
3 Area total calculada: 676.667
```

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.

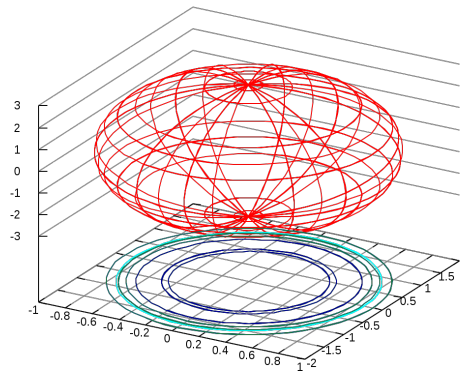


Imagen con propiedad de ©Wikipedia

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.

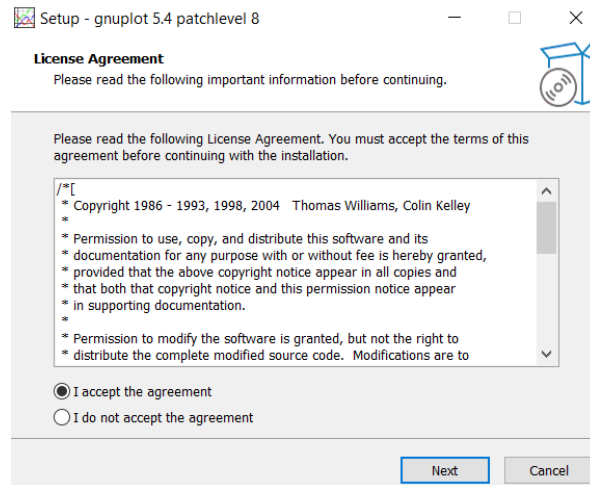


Imagen con propiedad de ©gnuplot

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 10: Generar InsertionSort.dat

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

Trapecioparalelo:

Listing 11: 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:

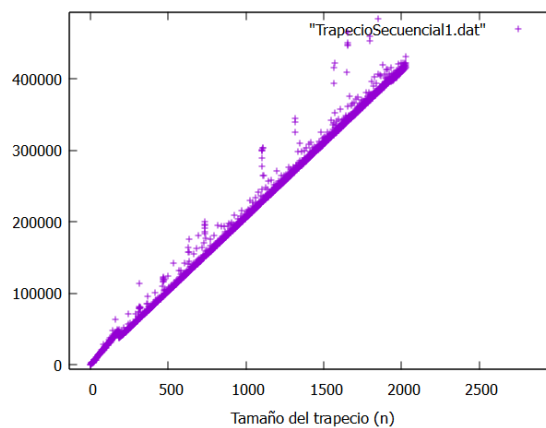
Una vez obtenido los .dat de cada ejecución de programa correspondiente, podemos utilizar el gnuplot para graficar e interpretar los gráficos correspondientes:

Listing 12: Cargar los datos a gnu plot

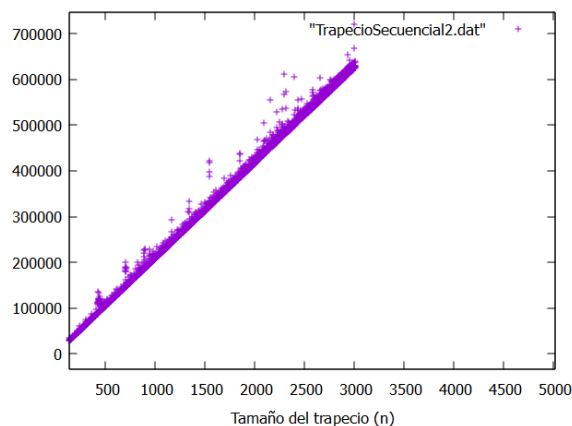
```
1 Terminal type is now 'qt'  
2 gnuplot> cd 'C:\lp3-24a\parcial\dat'  
3 gnuplot> set title "Análisis y comportamiento de paradigmas de programación"  
4 gnuplot> set xlabel "Tamaño del trapecio (n)"  
5 gnuplot> set ylabel "Tiempo de cálculo (nanosegundos)"  
6 gnuplot> plot "TrapecioSecuencial.dat"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

Salida del programa gnuplot:

Programación secuencial: mostrando diferentes casos



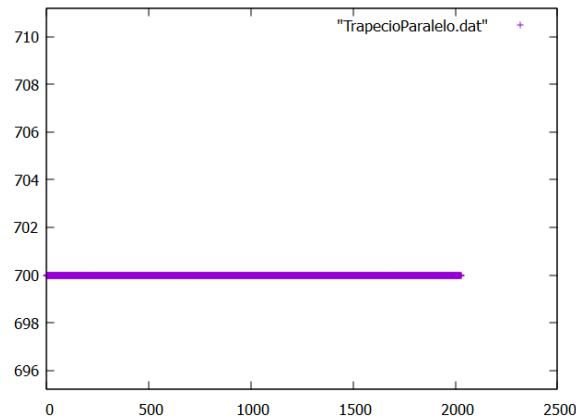
programación secuencial, límite inferior 0, límite superior=50, precisión 5, integral $2(x*x)+1$



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.

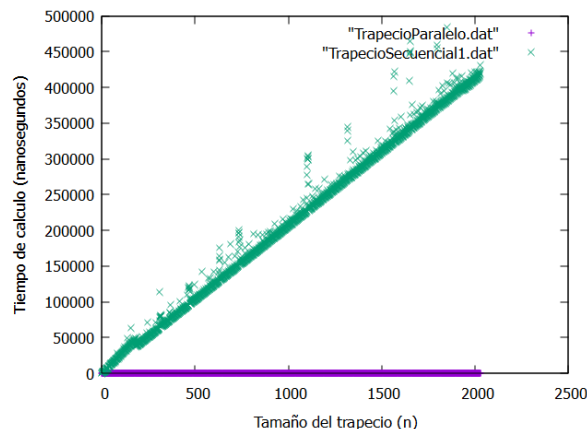
Programacion paralela: mostrando diferentes casos



programacion secuencial, limite inferior 0, limite superior=50, precision 5, integral $2(x*x)+1$

10.2.4. Analizar e interpretar los datos obtenidos:

Programacion 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			