



Las Americas Institute of Technology

Materia: Programación Paralela.

Título: Simulación de restaurante.

Integrantes Y Matrículas:

Víctor Raúl Alcántara Sánchez - 20231146

Cristopher Santana - 20231193

Raymond Isaac Moreno Guridis - 20230950

Elier Moreta Encarnación - 20231168

Lider: Pamela Blanco Vincent - 20231668

Carrera: Desarrollo de software.

Docente: Erick Leonardo Pérez Veloz.

Fecha:

23/04/2025

índice

1. Introducción	3
2. Descripción del Problema	4
3. Cumplimiento de los Requisitos del Proyecto	5
4. Diseño de la Solución	6
5. Implementación Técnica	8
6. Evaluación de Desempeño	14
7. Trabajo en Equipo.....	16
8. Conclusiones	17
9. Referencias	18
10. Anexos	19
Enlace al repositorio de Git (público) https://github.com/pame-12/Poyecto_Final_Programacion_Paralela.git	23

1. Introducción

Presentación general del proyecto

El proyecto en sí consiste en el desarrollo de una simulación que digamos modela el funcionamiento de un restaurante como tal, abarcando desde la preparación de platillos hasta la entrega de dichos pedidos. La simulación integra diversas etapas y tareas que se ejecutan en paralelo, permitiendo observar de manera dinámica y realista cómo se coordinan las operaciones internas (como la preparación en cocina) y el servicio de delivery.

Entre sus características se incluyen la aleatoriedad en la disposición del menú, la ejecución simultánea de múltiples pedidos y la implementación de mecanismos de sincronización (como locks) para el manejo de datos compartidos (por ejemplo, el total de pedidos y las estadísticas de entregas).

Asimismo, la simulación incorpora elementos de incertidumbre, como la posibilidad de entregas tardías y la política de ofrecer la comida de forma gratuita si el pedido se demora más de 30 minutos en situaciones específicas. Esta estructura permite evaluar el comportamiento del sistema a través de indicadores como el speedup y la eficiencia, y estudiar estrategias de paralelización y escalabilidad en contextos de alta carga.

Justificación del tema elegido

La elección de este tema se fundamenta en diversos aspectos de relevancia tanto teórica como práctica:

- **Relevancia técnica y educativa:**

El proyecto aborda conceptos fundamentales de la programación concurrente y paralela, permitiéndonos aplicar técnicas de gestión de hilos, sincronización y escalabilidad. Esto es crucial para entender cómo se diseñan sistemas robustos y eficientes en entornos que demandan alta.

- **Aplicación en entornos reales:**

La simulación se basa en una simple dinámica prácticamente es un restaurante real, donde la coordinación entre diferentes áreas (cocina, servicio al cliente y delivery) es esencial para garantizar la satisfacción del cliente. La inclusión de elementos aleatorios y de probabilidad (como los tiempos de entrega) refleja los desafíos reales en la gestión logística y operacional, permitiendo explorar soluciones que pueden ser aplicables en escenarios del mundo real.

- **Optimización de procesos y evaluación del rendimiento:**

Al incorporar métricas de rendimiento (speedup y eficiencia) y explorar distintos niveles de paralelización, el proyecto proporciona una plataforma para analizar y optimizar recursos en sistemas concurrentes.

Objetivos (general y específicos)

Objetivo General

Desarrollar una simulación de un restaurante que integre la preparación de platillos y el servicio de delivery, utilizando técnicas de programación concurrente y paralela para lograr una representación realista y escalable de las operaciones internas y logísticas.

Objetivos Específicos

- **Modelar y simular el flujo de trabajo de un restaurante:**
Definir y programar el proceso para la preparación de los platillos, estableciendo etapas claras (por ejemplo, preparación de ingredientes, cocción, emplatado, etc.) que se ejecuten de forma aleatoria y en paralelo.
- **Implementar mecanismos de ejecución concurrente:**
Utilizar hilos para simular la ejecución paralela de múltiples pedidos y platillos, permitiendo la coordinación simultánea entre la preparación en cocina y el envío del pedido.
- **Gestionar la sincronización de tareas compartidas:**
Emplear herramientas de sincronización (como locks) para el manejo seguro y eficiente de datos críticos, tales como el número total de pedidos, tiempos de entrega y resultados de rendimiento.
- **Evaluar el rendimiento y la escalabilidad del sistema:**
Aplicar métricas como speedup y eficiencia para analizar el impacto del paralelismo en la ejecución del sistema, y estudiar diferentes estrategias de escalamiento para optimizar el rendimiento bajo diversas cargas de trabajo.

2. Descripción del Problema

➤ Contexto del problema seleccionado

El problema se centra en simular el funcionamiento de un restaurante, donde se prepara la comida y se realizan entregas. La idea es imitar el proceso real de un restaurante como tal, desde que se cocina el platillo hasta que llega al cliente, incluyendo pasos de manera aleatoria y en función de distintos tiempos.

➤ Aplicación del problema en un escenario real

En un escenario real, este problema se puede ver reflejado en la operación de un restaurante que debe coordinar la cocina con el servicio de entrega. Cada pedido tiene varios platillos como tal los cuales se deben preparar al mismo tiempo, lo que permite simular el ajetreo del lugar con mucha demanda en si. Además, se añaden detalles como la posibilidad de que una entrega tarde y la política de dar la comida gratis si pasa un tiempo determinado, lo que ayuda a pensar en soluciones prácticas para mejorar el servicio.

➤ **Importancia del paralelismo en la solución**

El uso del paralelismo es muy importante en este proyecto, ya que nos permite que prácticamente varias tareas se realicen al mismo tiempo sin esperar a que termine la anterior. Esto significa que, mientras se prepara un platillo, otro puede estar en proceso de entrega. Esta forma de trabajar mejora el tiempo de respuesta del sistema y hace que la operación del restaurante sea mucho más eficiente.

3. Cumplimiento de los Requisitos del Proyecto

1. Ejecución simultánea de múltiples tareas

Se genera “x” cantidad de pedidos y se procesa cada platillo en paralelo. Mientras se prepara un platillo, el servicio de delivery puede comenzar a enviar la comida, lo cual simula múltiples tareas en paralelo.

2. Necesidad de compartir datos entre tareas

Las tareas deben acceder a estructuras comunes como el total de pedidos, entregas a tiempo y entregas tardías, utilizando mecanismos como locks.

3. Exploración de diferentes estrategias de paralelización

Descomposición de datos: Utilizamos descomposición de datos, ya que tomamos una lista de pedidos y la dividimos en partes independientes, y a cada pedido se le aplicó la misma lógica, de procesamiento y entrega.

Paralelismo de tareas: Utilizamos paralelismo de tareas ya que mientras se preparan los pedidos, otros hilos que simulan los repartidores van entregando los pedidos que están listos, simulando el funcionamiento real de un restaurante.

Sincronización de Recursos Compartidos: Utilizamos mecanismos de sincronización como lock, para asegurarnos de que solo un hilo acceda a una variable compartida a la vez y evitar errores.

Control de paralelismo: Utilizamos `MaxDegreeOfParallelism` para limitar el número máximo de procesadores.

4. Escalabilidad con más recursos

Se puede aumentar la cantidad de hilos utilizados para procesar los pedidos a medida que se disponga de más procesadores.

5. Métricas de evaluación del rendimiento

Tiempo: Medimos el tiempo total de cada ejecución, tanto de forma secuencial como de forma paralela con diferente número de procesadores.

Speedup: Calculamos el Speedup de cada ejecución paralela, para verificar qué tan rápido son en comparación con la ejecución secuencial.

Eficiencia: Calculamos la eficiencia para ver que tan bien se aprovechan los recursos de cada ejecución paralela.

Entregas a tiempo: Calculamos la cantidad de entregas a tiempo.

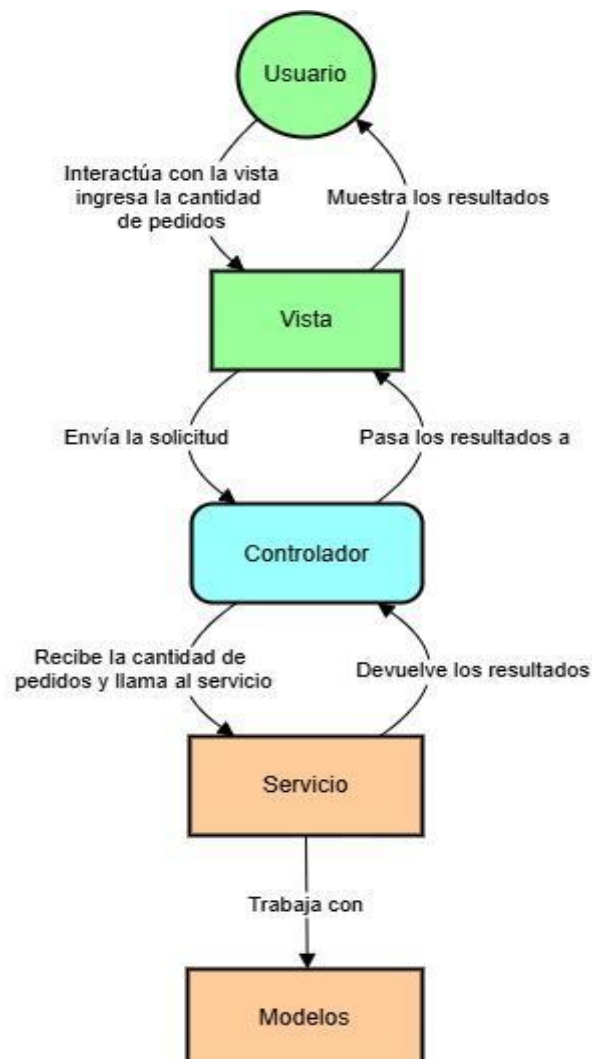
Entregas gratis: Calculamos la cantidad de entregas tardías, que salen gratis.

6. Aplicación a un problema del mundo real

Esta simulación refleja el funcionamiento de un restaurante real, con pedidos, platillos y tiempos de preparación, integrando aspectos como la posibilidad de retrasos y la probabilidad de entregas tardías.

4. Diseño de la Solución

➤ Arquitectura general del sistema



Vista ()

El usuario ingresa la cantidad de pedidos.

Controlador (RestauranteController)

Recibe la cantidad y llama al Servicio pasando ese valor.

Servicio (RestauranteService)

Genera los pedidos, ejecuta la simulación (secuencial y paralela), calcula métricas (como speedup y eficiencia), y retorna un el resultado.

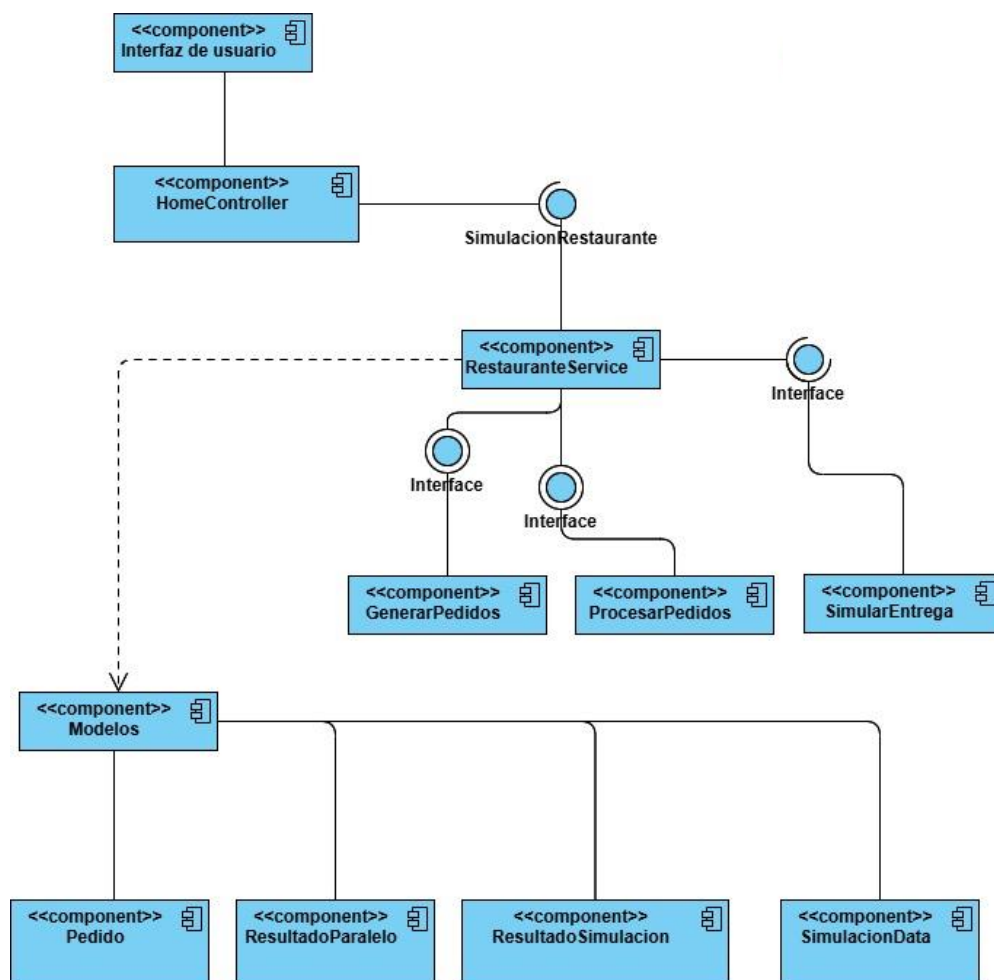
Controlador (RestauranteController)

Recibe el resultado y se lo pasa a la vista.

Vista()

Muestra los resultados.

➤ Diagrama de componentes/tareas paralelas



➤ Estrategia de paralelización utilizada

Descomposición de datos: Utilizamos descomposición de datos, ya que tomamos una lista de pedidos y la dividimos en partes independientes, y a cada pedido se le aplicó la misma lógica, de procesamiento y entrega.

Paralelismo de tareas: Utilizamos paralelismo de tareas ya que mientras se preparan los pedidos, otros hilos que simulan los repartidores van entregando los pedidos que están listos, simulando el funcionamiento real de un restaurante.

Sincronización de Recursos Compartidos: Utilizamos mecanismos de sincronización como lock, para asegurarnos de que solo un hilo acceda a una variable compartida a la vez y evitar errores.

Control de paralelismo: Utilizamos MaxDegreeOfParallelism para limitar el número máximo de procesadores.

➤ Herramientas y tecnologías empleadas

C# ASP.NET CORE MVC, HTML, CSS, BOOSTRAP, JS

5. Implementación Técnica

● Descripción de la estructura del proyecto

El sistema tiene una estructura de capas siguiendo el modelo de ASP.NET Core, dentro del src se encuentran las carpetas:

Modelos(Modelos): Son las clases que representan la información del sistema. En esta simulación:

Pedidos.cs: Almacena la información de cada pedido y posee:

Id: Identificador único de cada platillo.

Platillo: Nombre del platillo.

HoraPedido: Fecha y hora en la que se generó el pedido.

TiempoEntrega: Tiempo que tardó la entrega.

EsGratis: indica si el pedido es gratis si se retrasa.

ResultadoParalelo.cs: Almacena los resultados de las ejecuciones paralelas con diferente número de procesadores y posee:

Procesadores: Número de procesadores que se usaron en la ejecución.

Tiempo: Tiempo de ejecución.

Speedup: Cálculo del Speedup.

Eficiencia: Cálculo de la eficiencia.

EntregasATiempo::Pedidos entregados a tiempo.

EntregasGratis: Pedidos gratis por tardanza.

ResultadoSimulacion.cs: Almacena los resultados generales de la simulación.

TiempoSecuencial: Tiempo de la ejecución secuencial

EntregasATiempoSecuencial: Cantidad de pedidos entregados a tiempo en la ejecución secuencial.

EntregasGratisSecuencial: Cantidad de pedidos entregados gratis, por llegar tarde.

ResultadosParalelos: Es una lista que almacena los resultados para cada cantidad de procesador simulado.

TotalPedidos: Cantidad total de pedidos con la que se realizó la simulación.

EstadisticasPlatillos: Es un diccionario que almacena la cantidad de pedidos por platillo.

Servicios(Services): Almacena la lógica principal del sistema donde se hace la simulación.

RestauranteService.cs: contiene todos los métodos para la simulación del restaurante.

EjecutarSimulacion: Es el método principal que organiza la ejecución secuencial y paralela y calcula las métricas de rendimiento.

GenerarPedidos: Es el método que genera una lista inicial de los pedidos, le asigna un Id, un plato del menú de forma aleatoria y la hora en que se realizó el pedido.

ClonarPedidos: Hace una copia de la lista original de pedidos.

SimularEntrega: Simula la entrega de los pedidos, y calcula si llega a tiempo o es gratis.

ProcesarPedido: Simula la preparación de los pedidos.

Controladores(Controllers): Almacena el controlador que actúa como un puente entre la lógica y la interfaz.

RestauranteController.cs: Recibe las peticiones del usuario, llama al servicio y devuelve los resultados.

Vistas (Views): Son las páginas con la que el usuario interactúa.

➤ Explicación del código clave

El sistema está compuesto por un conjunto de métodos claves que hacen posible la simulación del restaurante dentro de los cuales se encuentran:

EjecutarSimulacion: Este método recibe por parámetro la cantidad de pedidos y una lista de platillos que simboliza el menú, que se usa para asignar aleatoriamente los platos a cada pedido.

Genera una lista de pedidos llamando al método **GenerarPedidos()**, donde a cada pedido se le asignan atributos como Id y plato.

Se creó una copia exacta de los pedidos originales llamando al método **ClonarPedidos()**, que recibe por parámetro la lista de pedidos originales y crea una copia exacta de los pedidos, para poder usar los mismos pedidos en ambas simulaciones.

Se creó una variable “resultado” la cual almacena los resultados de toda la simulación, tanto secuencial como paralela.

Para la ejecución secuencial utilizamos **foreach** (`var pedido in pedidosSecuencial`), que recorre la lista de pedidos y los procesa y entrega, uno a uno, primero llama al método **ProcesarPedido()** que simula la preparación de los platos y luego al método **SimularEntrega()** que simula la entrega.

Para la ejecución paralela se creó una lista llamada `nivelesParalelismo` para hacer la simulación con diferentes números de procesadores y la recorro con un `foreach`.

Se creó una cola segura llamada `pedidosListos` usando la clase **ConcurrentQueue<Pedido>()**, que permite que varios hilos puedan acceder a ella sin

provocar errores, en esta cola se almacenan los pedidos que ya estos preparados, para que los hilos encargados de la entrega puedan acceder a ella y se crea una variable llamada **pedidosPreparados**, que es como un contador de los pedidos listos.

Se usa un **Parallel.ForEach** para recorrer la lista de pedidos de forma paralela, limitando la el número máximo de procesadores al utilizar la variable opciones a la cual se le asigna el **MaxDegreeOfParallelism**, dentro de este se llama al método **ProcesarPedido** que simula el procesamiento del pedido de forma paralela y a medida que va terminando un pedido lo va agregando a la cola segura **pedidosListos.Enqueue(pedido)** usando **.Enqueue** para ir agregando cada pedido al final de la cola y de igual forma a medida que va terminando un pedido se va incrementando el contador **pedidosPreparados** usando lock para más seguridad.

Para la entrega se usa **Parallel.For** para ejecutar varios hilos al mismo tiempo desde 0 hasta el límite de procesadores de la iteración, dentro se usa un **while True** y se evalúan dos condiciones, **If (pedidosListos.TryDequeue(out var pedido))** si hay pedidos en la cola, lo saca y lo guarda en la variable **pedido** y simula la entrega llamando al método **SimularEntrega**, si no hay pedidos en la cola pasa al **else**, **if (pedidosPreparados == pedidosParalelo.Count && pedidosListos.IsEmpty)** donde se verifica si todos los pedidos han sido preparados y si la cola está vacía con **IsEmpty** que verifica si una cola está vacía o no, en caso positivo, termina el bucle con un **break** si no espera un poco con un **Thread.Sleep(10)** y se repite todo.

Al final se calculan todas las métricas, se guardan en la variable **resultado** y se retornan todos los resultados.

Para medir el tiempo de ejecución de las simulaciones, se utilizó la propiedad **Elapsed.TotalSeconds** en vez de **Elapsed.Milliseconds** para ver el tiempo en segundo.

Entrega de pedidos

En esta simulación de restaurante queríamos implementar la entrega de los pedidos y los organizamos de la siguiente manera con un random que tira número del 1 al 100 y así hacer las probabilidades:

- Entre 0 y 20: Se asigna un tiempo entre 5 y 10 minutos (20% de probabilidad)
- Entre 21 y 60: Se asigna un tiempo entre 10 y 15 minutos (40%)
- Entre 61 y 85: Se asigna un tiempo entre 15 y 20 minutos (25%)
- Entre 86 y 95: Se asigna un tiempo entre 20 y 25 minutos (10%)

- Mayor a 95: Se asigna un tiempo entre 31 y 40 minutos (5%) y se colocara como entrega gratis.

También agregue un contador usando un bloque Lock, si tenemos un tiempo mayor a 30, se incrementa el contador de entregas gratis y en el caso de que se entregó dentro del rango del precio se incrementa el contador de entregas a tiempos.

Implementación del frontend

Vista Principal

Objetivo

Permitir al usuario ingresar la cantidad de pedidos a simular y el menú o lista de platillos con los que se hará la simulación.

Vista de Resultados

Objetivo

Mostrar los resultados de la simulación tanto en forma secuencial como paralela.

Simulación Secuencial:

Tiempo total de procesamiento.

Cantidad de entregas a tiempo.

Entregas gratis (por tardanza).

Simulación Paralela

Muestra varias métricas agrupadas en gráficos circulares como

Speedup y eficiencia.

Entregas a tiempo y entregas gratis

➤ Uso de mecanismos de sincronización

lock: Se empleó para proteger variables compartidas como: pedidosPreparados, entregasGratis y entregasATiempo, para evitar que varios hilos accedan al mismo tiempo y evitar errores.

ConcurrentQueue<Pedido>: Es una cola segura, que maneja internamente su propio mecanismo de sincronización, está diseñada específicamente para trabajar con multihilos.

➤ **Justificación técnica de las decisiones tomadas**

A lo largo del desarrollo del proyecto de simulación de restaurante se tomaron algunas decisiones técnicas para cumplir con los requisitos del proyecto como:

Uso de TPL (Task Parallel Library): Se usaron herramientas como `Parallel.ForEach` y `Parallel.For` para manejar las tareas paralelas, ya que están diseñadas para trabajar con colecciones, dividen automáticamente el trabajo entre múltiples hilos y es fácil controlar el paralelismo con el uso de `MaxDegreeOfParallelism`.

Separación de preparación y entrega: En la parte de la ejecución paralela se separó la ejecución en dos etapas, para que ambas tanto el procesamiento de los pedidos como la entrega se ejecuten en paralelo, con una cola compartida, que permite iniciar la entrega tan pronto con un pedido esté listo, ya que de esa forma se puede simular el flujo real de un restaurante, donde a medida que van saliendo los pedidos se van entregando.

Uso de ConcurrentQueue<Pedido>: Se utilizó esta estructura de `ConcurrentQueue` para almacenar los pedidos preparados, ya que es una cola segura para trabajar hilos y no es necesario usar lock ya que esta estructura lo maneja internamente.

Sincronización con lock: Se utilizó lock para proteger variables compartidas como los contadores de entregas a tiempo o gratis y el de pedidos listos ya que asegura la integridad de los datos ya que evita que varios hilos modifiquen la misma variable a la vez.

6. Evaluación de Desempeño

➤ Comparativa entre ejecución secuencial y paralela.

La ejecución paralela mejoró significativamente el tiempo con respecto a la ejecución secuencial:

Con dos procesadores, el tiempo se reduce aproximadamente a la mitad y de igual forma va disminuyendo a medida que se le va aumentando el número de procesadores, con 4 procesadores el tiempo es casi 4 veces más rápido, con 6 procesadores es casi 6 veces más rápido y con 8 procesadores el tiempo es casi 8 veces más rápido que la ejecución secuencial.

La eficiencia en todas las ejecuciones Paralelas se mantiene por encima de 90, lo que indica que se están aprovechando bien los recursos cuando se hacen las tareas paralelas, lo que demuestra que la simulación puede escalar bien, si se implementa en hardware con múltiples hilos.

Tomando en cuenta los resultados, la ejecución con mejor rendimiento es la paralela con 8 procesadoras, ya que prácticamente reduce el tiempo 8 veces menos y tiene una alta eficiencia, lo cual es excelente ya que se están aprovechando bien los recursos.

➤ Métricas: tiempo de ejecución, eficiencia, escalabilidad.

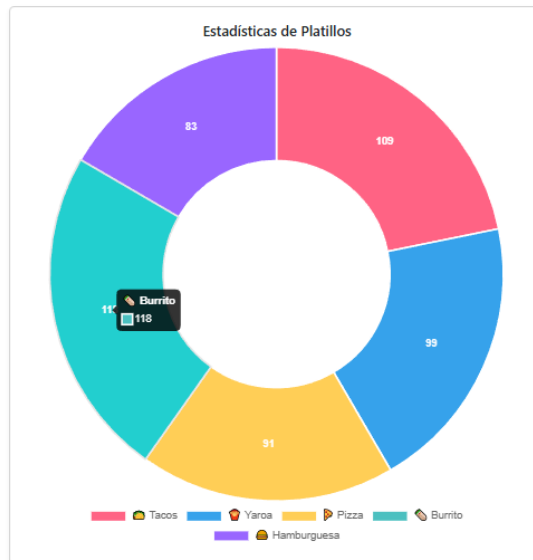
Simulación Secuencial					
Tiempo: 34.42 s					
Entregas a tiempo: 486					
Entregas gratis: 14					
Simulaciones Paralelas					
Procesadores	Tiempo (s)	Speedup	Eficiencia	Entregas a Tiempo	Entregas Gratis
2	17.31	1.99	99.5	468	32
4	8.76	3.93	98.25	476	24
6	5.88	5.85	97.5	481	19
8	4.5	7.65	95.62	486	14

➤ Gráficas o tablas con resultados.



Resultados de la Simulación

📊 Estadísticas de Platos



Total de Pedidos

500

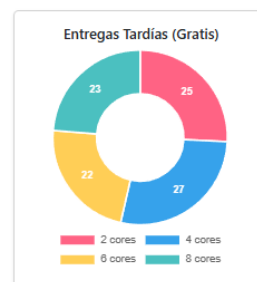
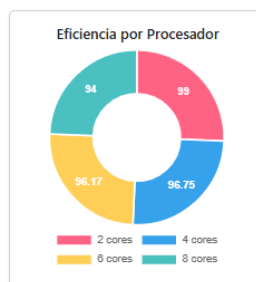
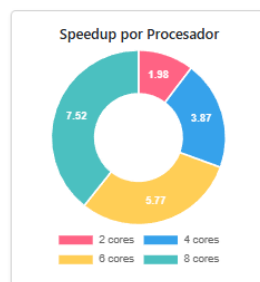
Simulación Secuencial

⌚ Tiempo: 34.44 s
📦 Entregas a tiempo: 477
📦 Entregas gratis: 23

🔄 Simulaciones Paralelas



Visualización Gráfica Circular



[Volver al inicio](#)

➤ Análisis de cuellos de botella o limitaciones.

Se puede generar un cuello de botella si hay muchos pedidos y no hay repartidores disponibles es decir procesadores.

7. Trabajo en Equipo

➤ Descripción del reparto de tareas

Frontend (Views, Controllers):

Elier Moreta Encarnación

Backend(Services, Models):

Raymond Isaac Moreno Guridis

Cristopher Alejandro Santana Peralta

Pamela Blanco Vincent

Documentación:

Víctor Raúl Alcántara Sánchez Y Pamela Blanco Vincent.

➤ Herramientas utilizadas para coordinación

Utilizamos Git Y Github

Git es un sistema de control de versiones distribuido que permite registrar de manera rápida y segura los cambios de tu código en repositorios locales y remotos, facilitando el trabajo en equipo y la gestión de diferentes versiones de un proyecto Git. GitHub es una plataforma en la nube para alojar repositorios Git, donde puedes compartir tu código, revisar cambios mediante pull requests y colaborar de forma organizada con otros desarrolladores Github Docs.

8. Conclusiones

➤ Principales aprendizajes técnicos

VictorRaul: Bueno, de mi parte como encargado de la documentación, yo tendría 2 principales de aprendizajes en sí y son 1- saber qué partes de los códigos son las más importantes o necesitan explicación a fondo, y 2- capacidad de detectar inconsistencias o áreas poco claras de este sistema.

ElierMoreta: Como responsable del desarrollo del frontend, mis principales aprendizajes fueron integrar correctamente tecnologías como Bootstrap y Chart.js para lograr una interfaz visualmente atractiva y funcional y Comprender cómo estructurar y comunicar visualmente los datos generados por la simulación de forma clara y responsiva para el usuario.

Cristopher Santana: En este proyecto he tenido la oportunidad de trabajar de manera organizada con un equipo, trabaje en el área de backend y pude aprender como hacer integraciones de lo que habíamos trabajado en la clase llevado a una aplicación web.

Pamela Blanco: A lo largo del desarrollo del proyecto, pude adquirir y reforzar conocimientos, y creo que uno de los más importantes fue el uso de ConcurrentQueue que es una estructura de datos que permite manejar múltiples tareas paralelas de forma segura, esto me ayudó mucho a poder implementar la lógica del manejo de tareas paralelas.

Raymond Moreno: Como parte de la creación de delivery, uno de mis aprendizajes fue como resolver de manera aleatoria y probabilidad y los rangos de tiempo para simular de la manera más realista posible otro de mis conocimientos fue mejorar lo que es el uso del lock para hacer los contadores de las entregas a tiempos y la parte gratuita cuando supera los 30 minutos.

➤ Retos enfrentados y superados

VictorRaul: De mi parte en este proyecto un reto que enfrente fue básicamente hacer esta documentación, está bien que algunos compañeros aportaron su parte en este documento, pero la verdad la documentación en sí para mí como desarrollador es tediosa y aburrida.

ElierMoreta: Uno de los mayores retos fue lograr que la interfaz no solo fuera funcional, sino también intuitiva y visualmente amigable. Implementar las gráficas dinámicamente con Chart.js, integrarlas dentro del entorno Razor (C#) y hacerlas compatibles con dispositivos móviles y web requirió múltiples pruebas. Sin embargo, el uso de Bootstrap facilitó bastante el diseño responsivo y limpio que buscábamos.

Pamela Blanco: Sin duda alguna creo que la parte más retadora de este proyecto fue la parte paralela, lograr que los pedidos se procesaran y entregaran en paralelo de forma correcta sin que generara errores. Al principio fue un poquito complicado, pero luego investigué y aprendí a usar algunas estructuras como ConcurrentQueue que me permitió manejar los pedidos de forma segura entre varias tareas, lo que me permitió pues completar la lógica del sistema de manera correcta.

Raymond Moreno: Creo que uno de los retos fue el manejo de los porcentajes que no sabía de qué manera hacerlo sin con “case” o la manera que lo hice que fue que la máquina elija un número random del 1 al 100 y de este modo convertirlo a porcentaje para los diferentes minutos. También uno de mis retos fue que los hilos incrementaron los mismos contadores simultáneamente que fue corregido con un lock(lockObj).

➤ Posibles mejoras o líneas futuras

Estuvimos debatiendo posibles mejoras, lo único que podríamos considerar como posible mejora es agregarle que el cliente pueda agregar ingredientes extras.

9. Referencias

➤ Fuentes bibliográficas, técnicas o académicas consultadas

<https://git-scm.com/> (sitio oficial de Git)

<https://docs.github.com/en/get-started/start-your-journey/about-github-and-git>
(documentación oficial de GitHub)

Visual Paradigm. (2025, 22 de abril). Visual Paradigm - Online Productivity Suite.

<https://online.visual-paradigm.com/>

[ConcurrentQueue<T> Class \(System.Collections.Concurrent\) | Microsoft Learn](#)

[ConcurrentQueue<T>.Enqueue\(T\) Method \(System.Collections.Concurrent\) | Microsoft Learn](#)

[ConcurrentQueue<T>.TryDequeue\(T\) Method \(System.Collections.Concurrent\) | Microsoft Learn](#)

[ConcurrentQueue<T>.IsEmpty Property \(System.Collections.Concurrent\) | Microsoft Learn](#)

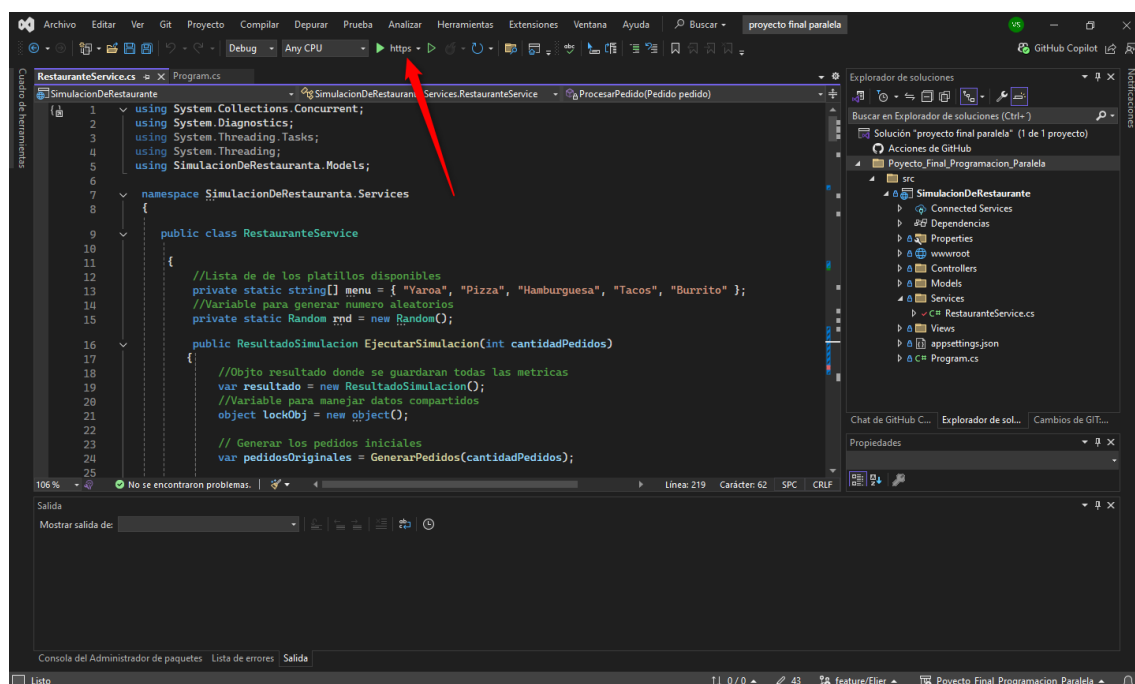
10. Anexos

➤ Manual de ejecución del sistema


Una vez hecho el GitClone del repositorio, darle doble click al archivo llamado “proyecto final paralela.sln”

Nombre	Fecha de modificación	Tipo	Tamaño
.vs	22/4/2025 5:51 p. m.	Carpeta de archivos	
Poyecto_Final_Programacion_Paralela	21/4/2025 5:10 p. m.	Carpeta de archivos	
proyecto final paralela.sln	20/4/2025 9:00 p. m.	Visual Studio Solu...	2 KB

Luego de que le des doble click a dicho archivo, solo sería esperar a que cargue y darle a la parte superior cerca del centro donde aparece un triángulo y a su derecha dice “https”



Una vez dentro seleccionas la cantidad de pedidos al igual de que platillos quieres en mi caso puse 500 pedidos y que sea aleatoriamente”seleccione todo” y le damos a iniciar simulación.



Bienvenido a la Simulación de Restaurante

Ingresar la cantidad de pedidos y selecciona uno o varios platillos del menú.

Cantidad de pedidos:

500

Selecciona los platillos del menú:

☒

Yaroa

☒

Pizza

☒

Hamburguesa

☒

Tacos

☒

Burrito

Iniciar Simulación

Ya solo sería ver que arroja en mi caso sería lo siguiente esto es basándome que mi pc tiene 4 procesadores arroja lo siguiente:

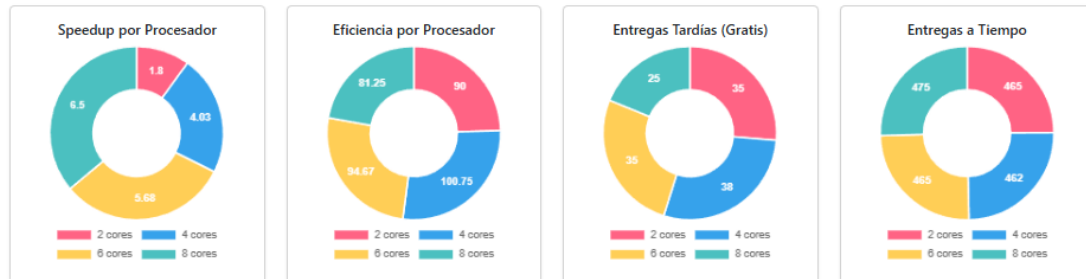


Simulación Secuencial

⌚ Tiempo: 36.92 s
📦 Entregas a tiempo: 478
📦 Entregas gratis: 22

🔧 Simulaciones Paralelas

📊 Visualización Gráfica Circular

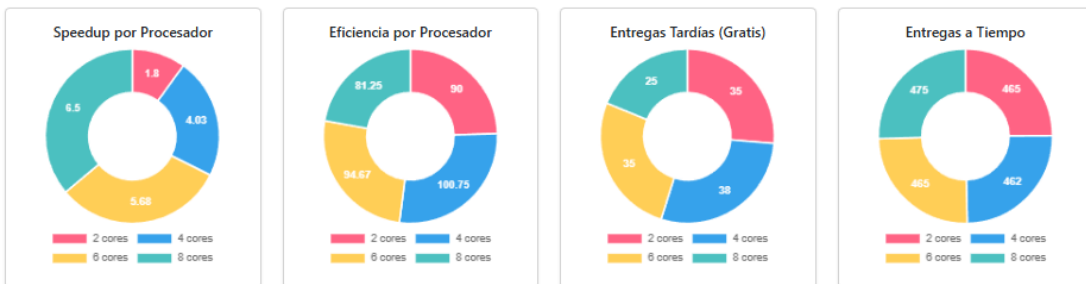


[Volver al inicio](#)

Para volver a ejecutar el programa solo es darle a “volver al inicio”

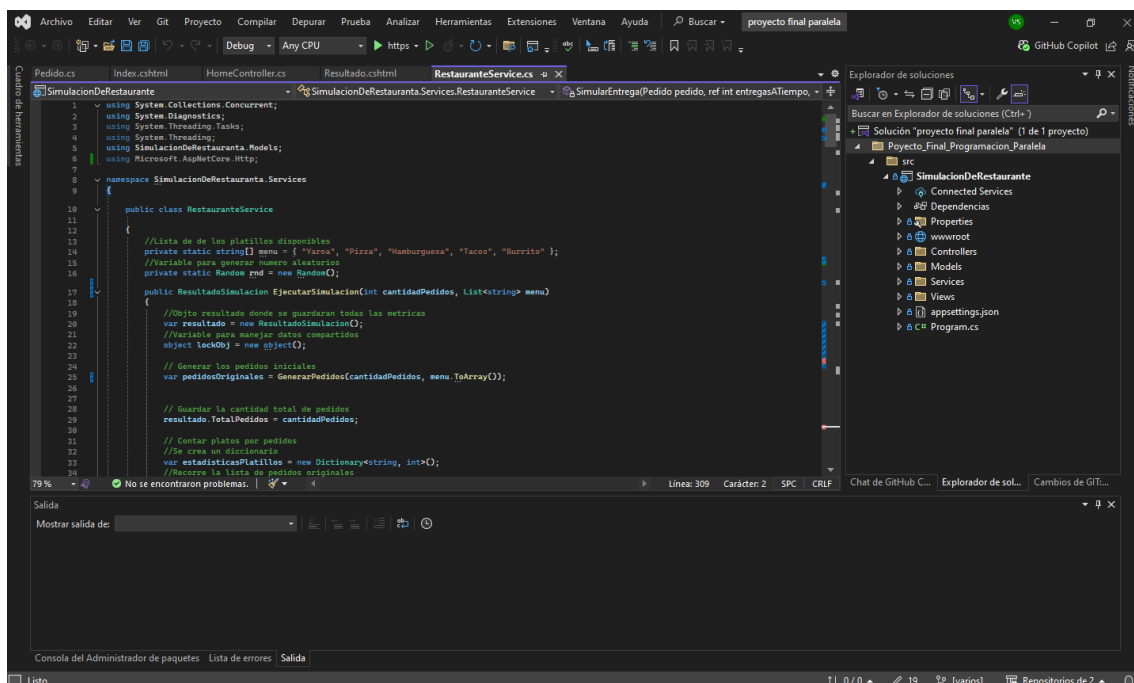
🔧 Simulaciones Paralelas

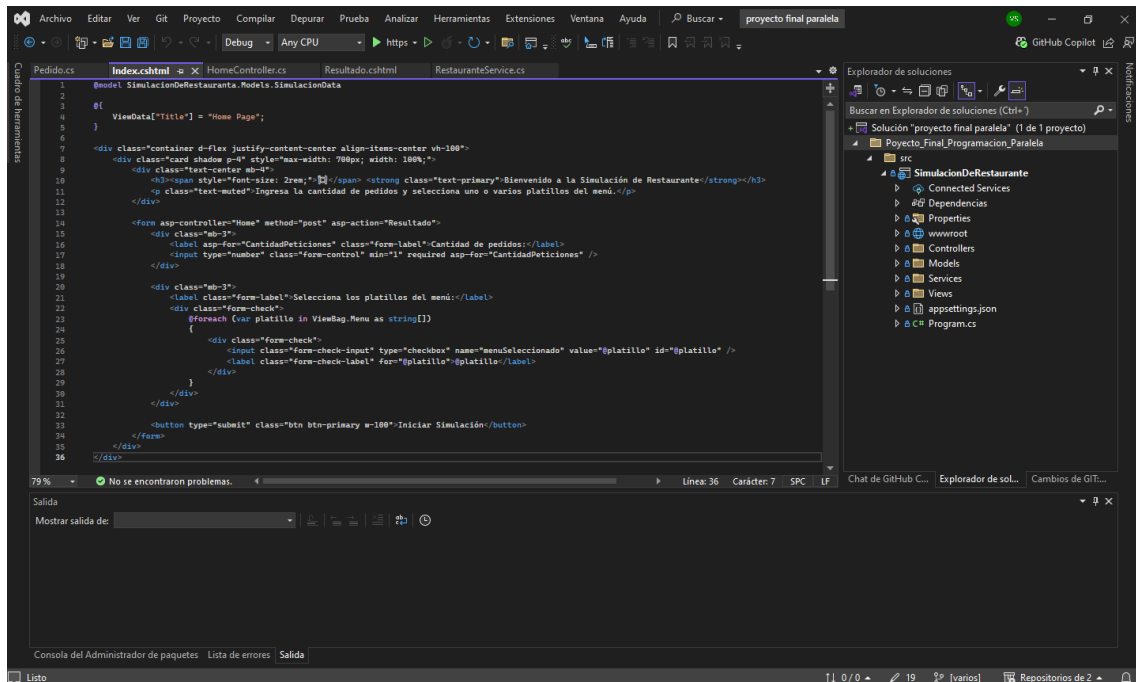
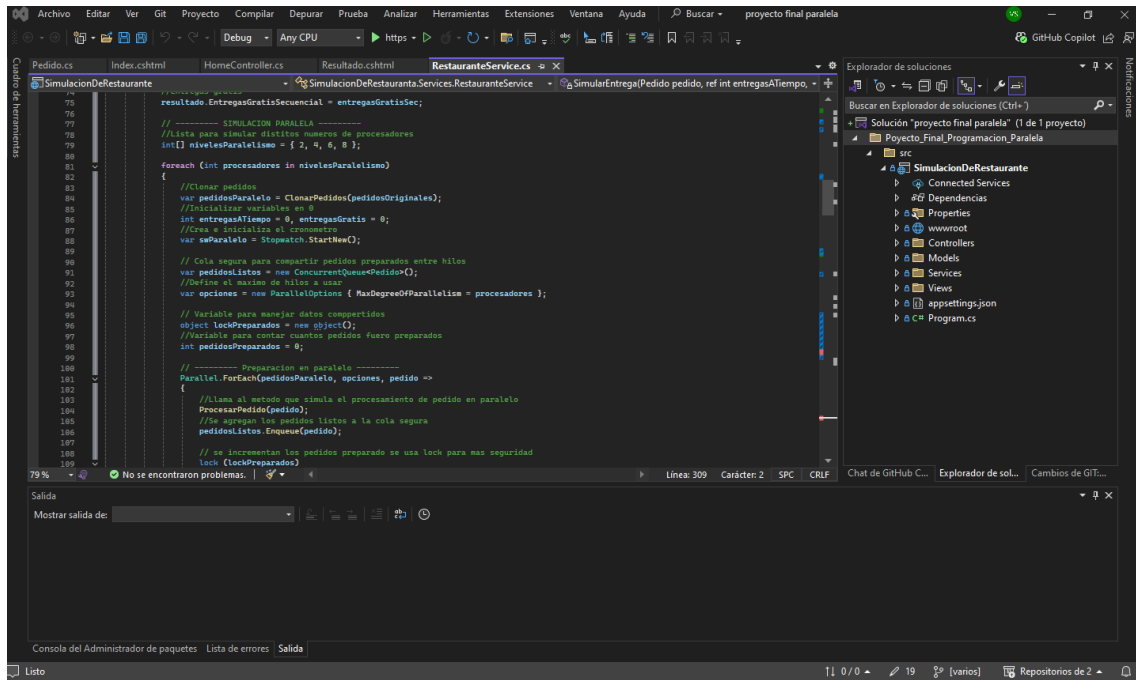
📊 Visualización Gráfica Circular

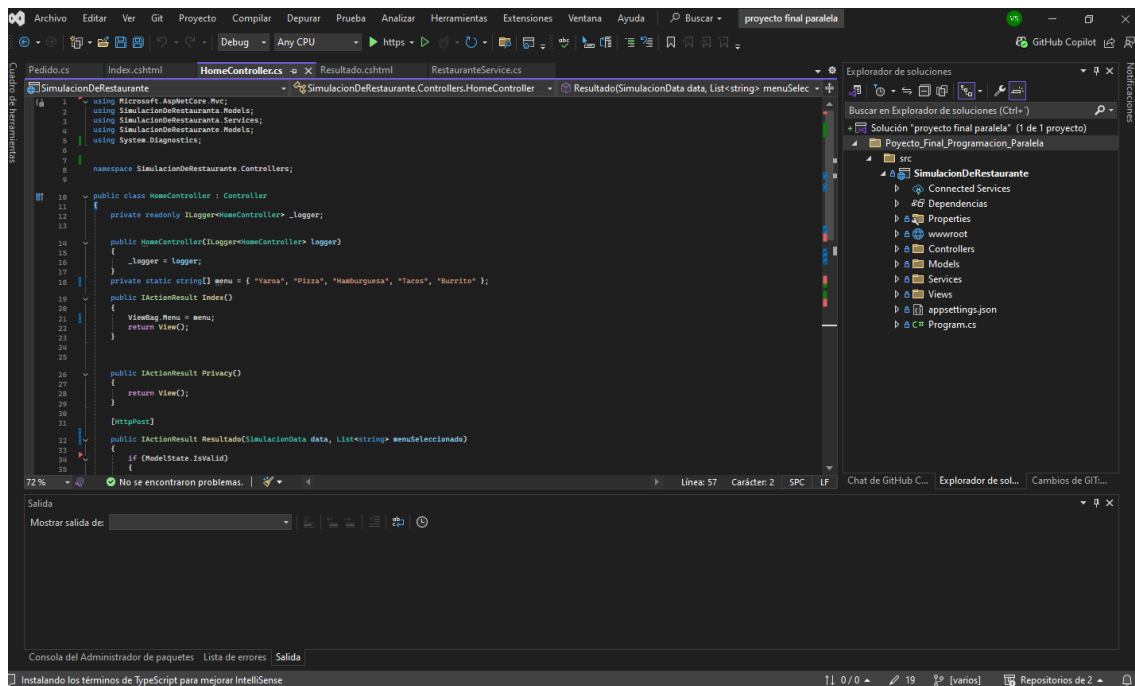


[Volver al inicio](#)

➤ Capturas adicionales, pruebas complementarias







Enlace al repositorio de Git (público) https://github.com/pame-12/Poyecto_Final_Programacion_Paralela.git

Nombre de usuarios en Github:

Pamela Blanco(pame-12)

Elier Moreta (Eliermoreta23)

Raymond Moreno (Ciyei0)

Cristopher (Casp06)

Victor R. (TDWTheMaster)