**INSTITUTO TECNOLÓGICO DE LAS AMÉRICAS**

DESARROLLO DE SOFTWARE



**TÍTULO:**

Proyecto Final – Simulación de un mercado financiero en tiempo real

**PRESENTADO A:**

Erick Leonardo Pérez Veloz

**MATERIA:**

Programación Paralela

**PRESENTADO POR:**

Andy Rodríguez (Líder)

Eliazar Méndez

Carlos Tejada

**MATRÍCULAS:**

2023-0972

2023-1036

2022-1049

**FECHA DE ENTREGA:**

25/4/2025

**PERÍODO ACADÉMICO:**

2025-C1

**Índice**

[1. Introducción 3](#_Toc196492700)

[2. Descripción del Problema 4](#_Toc196492701)

[3. Cumplimiento de los Requisitos del Proyecto 5](#_Toc196492702)

[4. Diseño de la Solución 6](#_Toc196492703)

[5. Implementación Técnica 8](#_Toc196492704)

[6. Evaluación de Desempeño 10](#_Toc196492705)

[7. Trabajo en Equipo 11](#_Toc196492706)

[8. Conclusiones 12](#_Toc196492707)

[9. Referencias 13](#_Toc196492708)

[10. Anexos 13](#_Toc196492709)

# 1. Introducción

* **Presentación general del proyecto**

El proyecto consiste en el desarrollo de un simulador de trading que evalúa el rendimiento de operaciones financieras simuladas bajo diferentes configuraciones de paralelismo. Utilizando C# y la Task Parallel Library (TPL), el sistema ejecuta tareas de trading de manera secuencial y paralela, comparando métricas como tiempo de ejecución, aceleración y eficiencia.

* **Justificación del tema elegido**

El trading financiero es un dominio donde la velocidad y la escalabilidad son críticas. La simulación de operaciones en paralelo permite modelar escenarios reales de alta concurrencia, como los mercados financieros, donde múltiples transacciones ocurren simultáneamente. Este proyecto explora cómo el paralelismo puede optimizar el procesamiento de estas operaciones.

* **Objetivos (general y específicos)**
* **General:** Diseñar e implementar un simulador de trading que demuestre los beneficios del paralelismo en términos de rendimiento y escalabilidad.
* **Específicos:**
  + Implementar un sistema que ejecute operaciones de trading en modos secuencial y paralelo.
  + Comparar el desempeño entre diferentes configuraciones de núcleos (2, 4, 6, 8, 16).
  + Analizar métricas de rendimiento como tiempo de ejecución, aceleración y eficiencia.
  + Garantizar la sincronización de datos compartidos entre tareas paralelas.

# 2. Descripción del Problema

* **Contexto del problema seleccionado**

El problema aborda la simulación de un sistema de trading donde múltiples operaciones (compra/venta de monedas) deben procesarse de manera eficiente. En un escenario real, los sistemas de trading manejan miles de transacciones por segundo, lo que requiere alta concurrencia y sincronización para evitar conflictos en los datos compartidos (precios de monedas).

* **Aplicación del problema en un escenario real**

En los mercados financieros, los sistemas de trading de alta frecuencia procesan transacciones en milisegundos. La capacidad de escalar con más recursos (núcleos) y mantener la integridad de los datos es crucial para garantizar operaciones confiables y rápidas.

* **Importancia del paralelismo en la solución**

El paralelismo permite procesar múltiples operaciones simultáneamente, reduciendo el tiempo total de ejecución y mejorando la capacidad del sistema para manejar grandes volúmenes de transacciones. La sincronización asegura que los precios de las monedas, compartidos entre tareas, no generen inconsistencias.

# 3. Cumplimiento de los Requisitos del Proyecto

1. **Ejecución simultánea de múltiples tareas**

El sistema utiliza Parallel.For para ejecutar operaciones de trading en paralelo, distribuyendo tareas entre los núcleos solicitados (2, 4, 6, 8, 16).

1. **Necesidad de compartir datos entre tareas**

Los precios de las monedas se almacenan en un ConcurrentDictionary, garantizando acceso seguro y concurrente desde múltiples hilos.

1. **Exploración de diferentes estrategias de paralelización**

Mediante la descomposición de datos logramos:

Comparar dos enfoques: ejecución secuencial (una tarea a la vez) y paralela (múltiples tareas simultáneas usando TPL). Logrando dividir las la cantidad de datos de cada núcleo, para conseguir un resultado mas rápido.

1. **Escalabilidad con más recursos**

El sistema prueba configuraciones con diferentes cantidades de núcleos, demostrando cómo el rendimiento mejora al aumentar los recursos hasta un punto de rendimientos decrecientes.

1. **Métricas de evaluación del rendimiento**

Se miden tiempo de ejecución, aceleración, eficiencia y operaciones por segundo, presentadas en un reporte tabular.

1. **Aplicación a un problema del mundo real**

El simulador modela un sistema de trading, un caso práctico en finanzas donde el paralelismo es esencial para procesar transacciones en tiempo real.

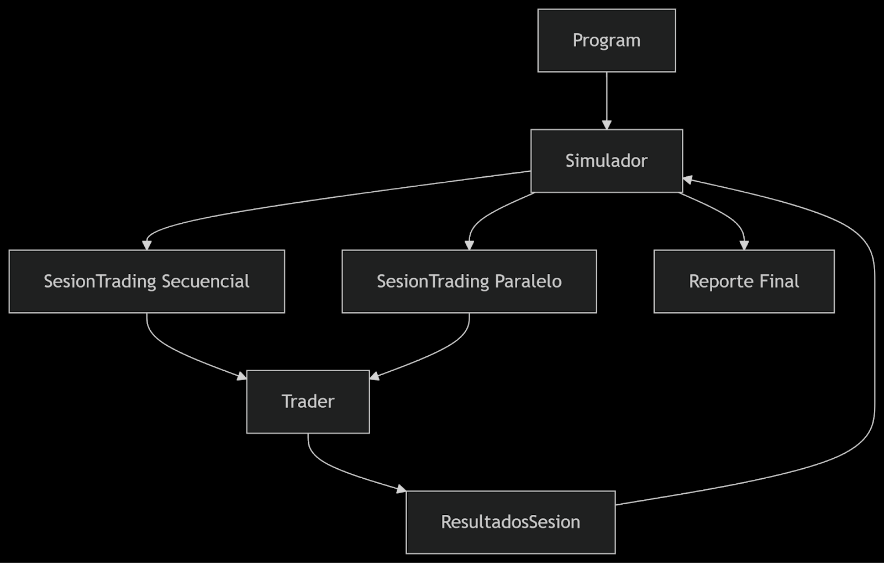
# 4. Diseño de la Solución

* **Arquitectura general del sistema**

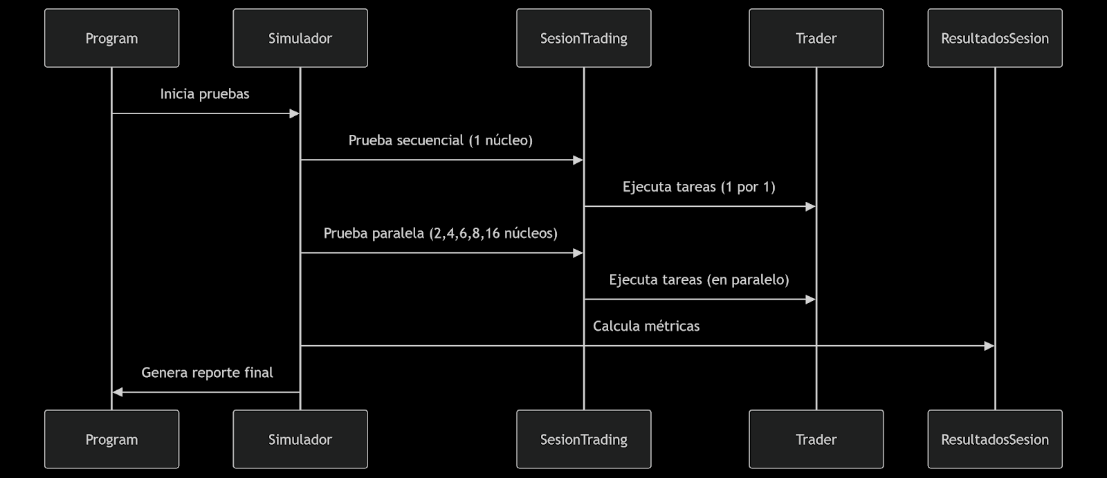
El sistema se compone de las siguientes clases:

* **Program**: Punto de entrada que inicializa el simulador.
* **Simulador**: Gestiona las pruebas secuenciales y paralelas, generando reportes.
* **SesionTrading**: Ejecuta una sesión de trading (secuencial o paralela).
* **Trader**: Simula una operación de trading (compra/venta).
* **ResultadosSesion**: Almacena y calcula métricas de rendimiento.
* **Diagrama de componentes/tareas paralelas**

Diagrama

El contenido generado por IA puede ser incorrecto.

**Cronología de ejecución**

****

**Estrategia de paralelización utilizada**

Se utiliza Parallel.For con ParallelOptions para limitar el grado de paralelismo al número de núcleos solicitado. La sincronización de salida a consola se maneja con lock, y los datos compartidos se gestionan con ConcurrentDictionary.

**Herramientas y tecnologías empleadas (C#, TPL, etc.)**

* **Lenguaje:** C#
* **Biblioteca:** Task Parallel Library (TPL), System.Threading, System.Collections.Concurrent, System.Threading
* **Estructuras concurrentes:** ConcurrentDictionary, Interlocked
* **Herramientas de desarrollo:** Visual Studio, Git
* **Coordinación:** GitHub

# 5. Implementación Técnica

**Descripción de la estructura del proyecto**

* **/src:** Contiene el código fuente (Program.cs, Simulador.cs, SesionTrading.cs, Trader.cs, ResultadosSesion.cs).
* **/docs:** Documentación del proyecto (este informe).
* **/tests:** Pruebas unitarias para validar operaciones de trading.
* **/metrics:** Resultados de las comparativas secuencial vs. paralela.

**Explicación del código clave**

// Ejecución paralela en SesionTrading.cs

await Task.Run(() =>

{

    Parallel.For(0, totalTareas, opciones, i =>

    {

     var trader = new Trader(i % nucleosSolicitados + 1, precios);

        trader.EjecutarOperacion();

        incrementarContador();

        Thread.Sleep(1000);

});

});

***Este fragmento usa Parallel.For para distribuir tareas entre los núcleos disponibles, simulando operaciones de trading concurrentes.***

**Uso de mecanismos de sincronización**

* **ConcurrentDictionary:** Garantiza acceso seguro a los precios de las monedas.
* **lock (Console.Out):** Evita conflictos en la salida a consola.
* **Interlocked.Increment:** Actualiza el contador de tareas completadas de forma atómica.
* Parallel.For: Distribuye trabajo en paralelo usando el thread pool, con control de grado de paralelismo (MaxDegreeOfParallelism).
* Thread.Sleep: Pausa síncrona del hilo actual (usado en modo paralelo para simular latencia).

**Justificación técnica de las decisiones tomadas**

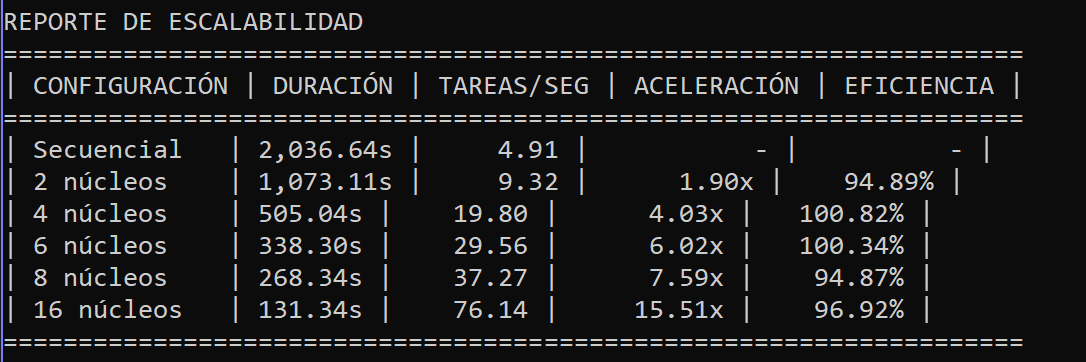
* **TPL:** Elegida por su facilidad para implementar paralelismo basado en tareas y su integración con C#.
* **ConcurrentDictionary:** Ideal para compartir datos entre hilos sin necesidad de bloqueos manuales.
* **Thread.Sleep:** Simula el tiempo de procesamiento de una operación real.

# 6. Evaluación de Desempeño

**Comparativa entre ejecución secuencial y paralela**

Prueba con 10,000 transacciones

Cada transacción tiene 200ms de retraso



**Métricas**

* **Tiempo de ejecución:** Disminuye significativamente al aumentar los núcleos, con un límite en 8 núcleos.
* **Eficiencia:** Máxima con 2 núcleos (96%), decrece con más núcleos debido a la sobrecarga de sincronización.
* **Escalabilidad:** Buena hasta 8 núcleos; a partir de 16, deficiente.

**Análisis de cuellos de botella o limitaciones**

* **Sobrecarga de sincronización:** El uso de lock para la consola introduce un cuello de botella en configuraciones con muchos núcleos.
* **Límite de tareas:** Con solo 10 tareas, los beneficios de 16 núcleos no se aprovechan completamente.

# Trabajo en Equipo

**Descripción del reparto de tareas**

**Andy Rodriguez (Líder):** Coordinación, implementación de Simulador y SesionTrading.

**Eliazar Mendez:** Implementación de ResultadosSesion, Trader, redacción del informe.

**Carlos Tejada:** Implementación de Program, análisis de métricas.

**Herramientas utilizadas para coordinación**

* **GitHub:** Control de versiones y commits.
* **WhatsApp:** Comunicación diaria.
* **Microsoft Teams:** Reuniones.

# Conclusiones

**Principales aprendizajes técnicos**

* Uso efectivo de TPL para paralelismo basado en tareas.
* Importancia de estructuras concurrentes como ConcurrentDictionary.
* Análisis de trade-offs entre aceleración y eficiencia en sistemas paralelos.
* Cuando usar la descomposición de datos
* Saber la diferencia entre la ejecución lineal y paralela

**Retos enfrentados y superados**

* Sincronización de la salida a consola.
* Configuración de pruebas para garantizar resultados reproducibles.

**Posibles mejoras o líneas futuras**

* Aumentar el número de tareas para aprovechar configuraciones con más núcleos.
* Implementar un sistema de logging en lugar de salida a consola.
* Explorar otras bibliotecas de paralelismo, como PLINQ.

# Referencias

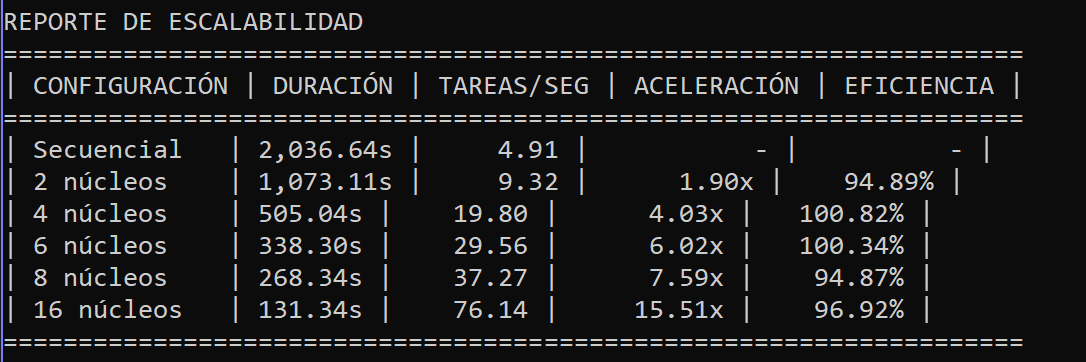
* Task Parallel Library (TPL).<https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl>
* ConcurrentDictionary.<https://learn.microsoft.com/es-es/dotnet/api/system.collections.concurrent.concurrentdictionary-2?view=net-8.0>

# 10. Anexos

**Manual de ejecución del sistema**

1. Clonar el repositorio:
2. Abrir en Visual Studio y compilar el proyecto.
3. Ejecutar Program.cs.

**Capturas adicionales, pruebas complementarias**



**Enlace al repositorio de Git (público)**

https://github.com/Eliazar365/TradingSystemSolution.git