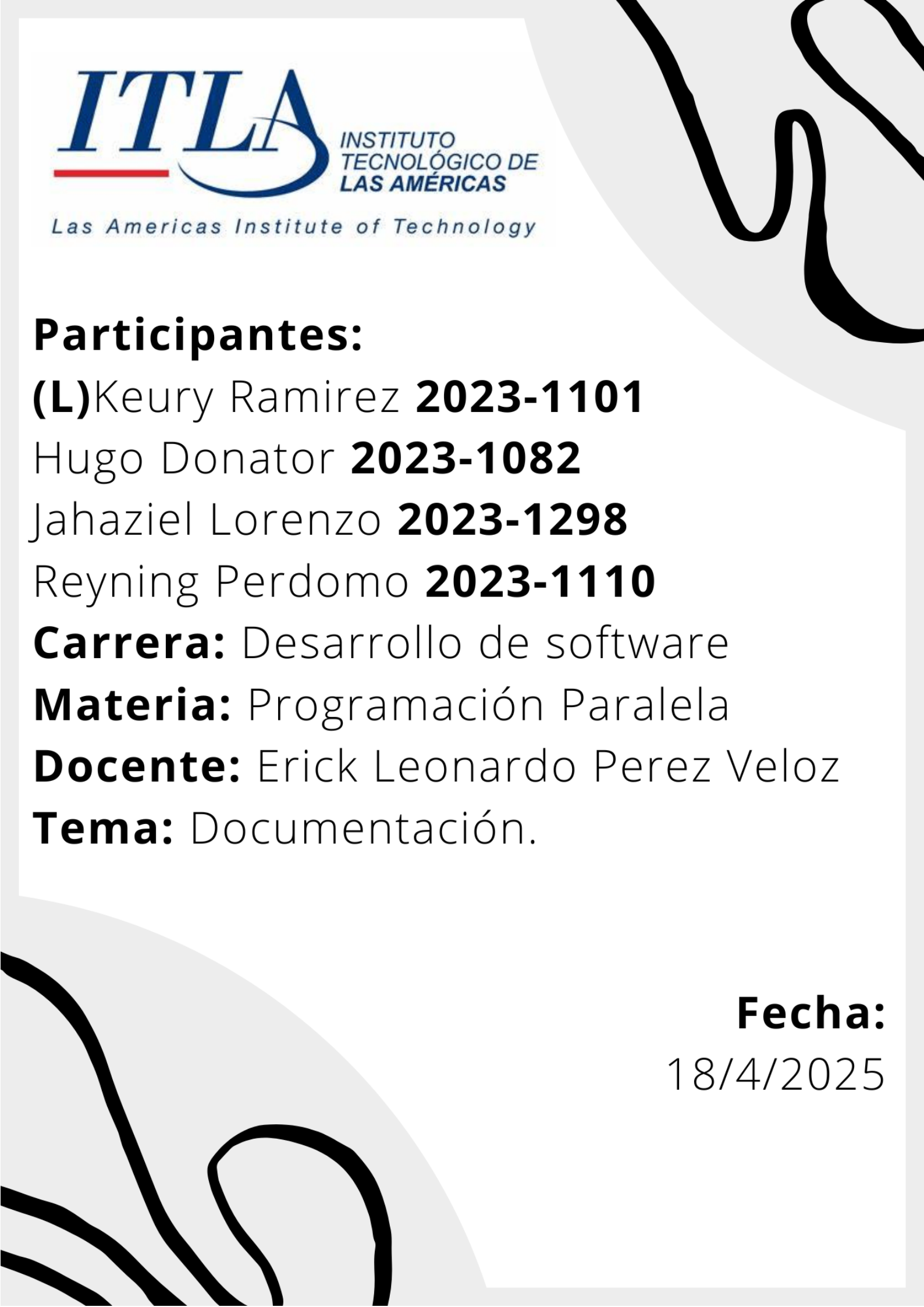


Participantes:
(L) Keury Ramirez **2023-1101**
Hugo Donator **2023-1082**
Jahaziel Lorenzo **2023-1298**
Reyning Perdomo **2023-1110**
Carrera: Desarrollo de software
Materia: Programación Paralela
Docente: Erick Leonardo Perez Veloz
Tema: Documentación.

Fecha:
18/4/2025

Tabla de contenido

Mercado Bursátil

Introducción

Presentación general del proyecto

Este proyecto desarrolla un simulador de mercado bursátil que reproduce de forma paralela la dinámica de compra y venta de acciones mediante agentes independientes, gestionando de manera segura el acceso a los datos compartidos y ofreciendo la posibilidad de comparar distintas estrategias de paralelización y medir su rendimiento con métricas precisas analizando los datos obtenidos del programa para la realización de gráficos que nos ayudarán a entender mejor los datos del proyecto.

Justificación del tema elegido

Elegimos este tema porque refleja uno de los desafíos más críticos en la computación moderna: aprovechar el **paralelismo** para procesar grandes volúmenes de operaciones en tiempo real, tal como ocurre en las bolsas de valores, plataformas de criptomonedas y sistemas de subastas en línea. Además, permite experimentar con técnicas de **sincronización** y **escalabilidad**, habilidades clave en el desarrollo de software de alto rendimiento y en infraestructuras distribuidas.

Objetivos

Objetivo general:

Desarrollar un sistema de simulación que reproduzca el funcionamiento de un mercado bursátil, permitiendo medir y comparar el desempeño entre ejecuciones secuenciales y paralelas, utilizando estrategias de paralelización eficientes y aprovechando al máximo los recursos del procesador.

Objetivos específicos:

- Implementar agentes como tareas independientes usando async/await y Parallel.ForEachAsync para lograr una ejecución eficiente y escalable.
- Obtener precios de acciones de forma asíncrona desde Yahoo Finance, garantizando integridad de datos.
- Optimizar el uso del procesador distribuyendo la carga entre los núcleos (MaxDegreeOfParallelism).
- Medir y reportar el rendimiento: tiempo total, operaciones por segundo, eficiencia y speedup.
- Comparar ejecuciones secuenciales y paralelas para evaluar la ganancia de rendimiento.

Descripción del Problema

En los entornos financieros reales, el mercado bursátil es altamente dinámico, miles de transacciones ocurren al mismo tiempo, influenciando constantemente el valor de las acciones. Modelar y entender este comportamiento simultáneo es fundamental para estudiar el impacto de la concurrencia en los sistemas de trading y evaluar cómo responden ante cargas de trabajo paralelas. Sin embargo, desarrollar simuladores que repliquen estas condiciones de forma eficiente, midiendo además el rendimiento bajo distintas estrategias de ejecución (secuencial vs paralela), sigue siendo un reto técnico importante.

Nuestro sistema busca abordar precisamente esta necesidad: crear un simulador que permita reproducir, analizar y comparar la ejecución de múltiples agentes bursátiles de forma escalable, eficiente y segura, gestionando adecuadamente la concurrencia y maximizando el aprovechamiento de los recursos computacionales disponibles.

Contexto del problema seleccionado

Imaginemos un mercado bursátil en pleno funcionamiento, donde cientos o miles de inversores realizan operaciones de compra y venta simultáneamente. Para representar este fenómeno, el sistema crea múltiples "agentes" virtuales, cada uno como una tarea asíncrona independiente, que consulta precios en tiempo real a través de la API de Yahoo Finance y decide operaciones de compra o venta.

Dado que varios agentes pueden intentar modificar o leer la misma información simultáneamente, es crucial implementar mecanismos de sincronización que garanticen la integridad de los datos y eviten condiciones de carrera. Además, el sistema busca medir el impacto de ejecutar estas operaciones de manera secuencial frente a una ejecución paralela, evaluando el rendimiento, la eficiencia y la escalabilidad que se puede lograr al aprovechar múltiples núcleos de procesamiento.

Aplicación del problema en un escenario real

En mercados bursátiles reales, miles de operaciones de compra y venta de acciones ocurren de manera simultánea. Para que estos sistemas funcionen correctamente, deben garantizar acceso rápido, seguro y consistente a la información crítica, como los precios de las acciones. Nuestro simulador replica esta dinámica creando agentes que consultan y actualizan precios en paralelo, permitiendo medir el impacto de la concurrencia en el rendimiento del sistema.

Esta simulación ofrece una base práctica para entender y diseñar sistemas escalables y eficientes, aplicables en plataformas de trading, motores de órdenes y servicios de monitoreo bursátil en tiempo real.

Importancia del paralelismo en la solución

- **Velocidad y realismo:** Ejecutar agentes en paralelo mejora la simulación, reflejando el comportamiento de un mercado real donde las transacciones son simultáneas.
- **Escalabilidad:** El sistema puede escalar fácilmente para manejar más agentes o núcleos, aprovechando los recursos disponibles sin reescribir la lógica.
- **Comparación de enfoques:** Probar diferentes métodos de paralelización nos ayuda a elegir la opción más eficiente para cada caso.
- **Medición del rendimiento:** Monitorear métricas clave permite identificar cuellos de botella y optimizar la simulación para mejorar velocidad y precisión.

Cumplimiento de los Requisitos del Proyecto

Este proyecto consiste en simular el comportamiento de un mercado bursátil en el que varios agentes realizan operaciones de compra y venta de acciones de manera paralela.

Ejecución simultánea de múltiples tareas

El simulador utiliza tareas paralelas (por ejemplo, Task.Run(), Parallel.For) para permitir que los agentes ejecuten operaciones de compra/venta de acciones simultáneamente. Cada agente es representado por una tarea que se ejecuta de manera independiente, simulando un mercado donde las transacciones ocurren al mismo tiempo.

Necesidad de compartir datos entre tareas

Las tareas (agentes) necesitan acceder a los precios compartidos de las acciones en el objeto Mercado. Para mantener la consistencia de estos precios, se debe implementar una sincronización de datos, como el uso de lock o estructuras de datos concurrentes, asegurando que los agentes no modifiquen los precios simultáneamente de manera inconsistente.

Permitir la exploración de diferentes estrategias de paralelización

El proyecto permite comparar diferentes estrategias de paralelización, como la ejecución mediante Parallel.For, el uso de para cada agente, o incluso paralelización mediante técnicas de divide y vencerás (como particionar la lista de acciones).

Tener la capacidad de escalar con más recursos

El simulador es escalable. Si se aumenta el número de agentes o núcleos de CPU disponibles, el sistema puede ejecutar más tareas en paralelo sin perder rendimiento. Además, el sistema puede escalar horizontalmente, distribuyendo la carga a través de múltiples servidores o contenedores si se implementa en la nube.

Métricas de evaluación del rendimiento

Tiempo total de simulación:

- ***Secuencial*:** 1089 ms
- ***Paralelo*:** 233 ms

La comparación entre la ejecución secuencial y paralela muestra una mejora significativa en el tiempo de simulación al usar paralelismo.

Operaciones realizadas por segundo:

Se procesaron 1000 operaciones en total, lo que ayuda a evaluar el rendimiento del sistema en función de la cantidad de tareas realizadas por unidad de tiempo.

Rendimiento por agente:

El rendimiento por agente es optimizado con la paralelización, permitiendo que cada tarea (agente) se ejecute de manera independiente y eficiente.

Eficiencia de la paralelización:

- ***Speedup*:** 4.67x
- ***Eficiencia*:** 58.42 %

El *speedup* muestra que la simulación paralela fue 4.67 veces más rápida que la secuencial. Sin embargo, la eficiencia no es perfecta (58.42%), lo que indica que hay margen para mejorar la distribución de las tareas.

Información del sistema:

- ***Procesadores lógicos*:** 8
- ***Hilos usados*:** 29

La simulación aprovechó los recursos del sistema utilizando múltiples hilos, distribuidos entre los núcleos lógicos disponibles.

Gráficos exportados:

Los gráficos detallados del rendimiento pueden encontrarse en la carpeta 'MetricsExport' para un análisis visual más profundo.

Aplicación a un problema del mundo real

Este simulador se puede aplicar directamente al desarrollo y prueba de sistemas de trading algorítmico en entornos reales, donde es vital procesar miles de órdenes simultáneas sin perder consistencia ni velocidad. Por ejemplo, una firma de inversión cuantitativa puede usarlo para evaluar distintas estrategias de paralelización y sincronización antes de desplegarlas en producción, asegurando que sus algoritmos de compra/venta reaccionen correctamente a cambios de precio en tiempo real, asimismo, plataformas de criptomonedas o bolsas de valores pueden emplear esta herramienta para medir la capacidad de sus infraestructuras al escalar en **núcleos** de **CPU** o **servidores** adicionales, detectar **cuellos de botel**la y **optimizar recursos** antes de enfrentar operaciones con auténtico dinero y clientes.

Diseño de la Solución

La aplicación arranca en **Program.cs**, donde se desarrolla todo:

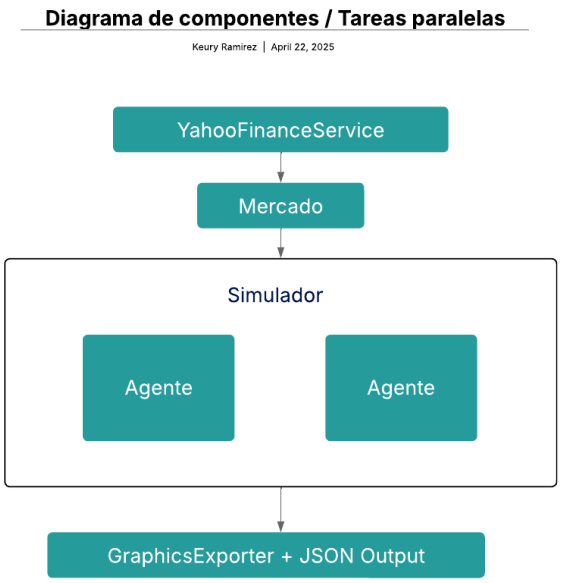
- **Obtención de datos:** un servicio se conecta a Yahoo Finance para traer los precios.
- **Modelo de mercado:** se mantiene un "Mercado" con todas las acciones, cuyos precios pueden actualizarse.
- **Agentes simulados:** varios bots ("Agente1", "Agente2", ...) que, de forma repetitiva, eligen una acción al azar, la suben o bajan de precio y envían esa operación a un "broker" simulado.
- **Simulador:** coordina a los agentes, primero corriéndolos de uno en uno (modo secuencial) y luego todos a la vez (modo paralelo), mide tiempos y calcula métricas.
- **Exportación de resultados:** al final se generan archivos JSON y gráficos (con ScottPlot) para visualizar comparaciones de rendimiento y recursos del sistema.

Arquitectura general del sistema

Todo está conectado de forma muy lineal en el programa principal, pero con cada pieza bien separada para que sea escalable, fácil de entender y mantener.

- **Capa de servicios:**
  - YahooFinanceService pide y procesa precios desde la API.
- **Capa de dominio:**
  - Accion y Mercado definen el modelo de datos y su gestión concurrente.
  - SimulatedBrokerClient formatea y muestra las órdenes.
- **Capa de agentes y simulación:**
  - Agente toma decisiones aleatorias y opera sobre el mercado.
  - Simulador orquesta la ejecución, mide tiempos y calcula métricas.
- **Capa de presentación/exportación:**
  - GraphicsExporter crea gráficas y guarda un JSON con las métricas.

Diagrama de componentes / Tareas paralelas



- En **modo secuencial**, cada agente corre uno tras otro.
- En **modo paralelo**, todos los agentes se lanzan al mismo tiempo usando un Parallel.ForEachAsync, limitado al número de procesadores lógicos.

Estrategia de paralelización utilizada

La estrategia de paralelización implementada en el sistema consistió en comparar la ejecución secuencial tradicional contra un esquema de procesamiento concurrente. Primero, se midió el tiempo total de ejecución ejecutando cada agente de manera individual mediante un bucle for y await agente.EjecutarAsync(). Luego, se aplicó Parallel.ForEachAsync, configurando la concurrencia máxima al número de procesadores disponibles (Environment.ProcessorCount), permitiendo así que múltiples agentes se ejecutaran simultáneamente y optimizando el uso del hardware.

La evaluación de resultados incluyó el cálculo de dos métricas fundamentales:

- **Speedup:** definido como la razón entre el tiempo de ejecución secuencial y el tiempo de ejecución paralela (Speedup = tiempoSecuencial / tiempoParalelo).
- **Eficiencia:** determinada como el cociente entre el speedup obtenido y el número total de procesadores utilizados (Eficiencia = speedup / número de procesadores).

Herramientas y tecnologías empleadas

- **Lenguaje:** C# con .NET (uso de async/await, Parallel.ForEachAsync, CancellationToken).
- **TPL (Task Parallel Library):** para gestionar la ejecución en paralelo y controlar el grado de simultaneidad.
- **ConcurrentDictionary:** en Mercado, para asegurar que varias hilos puedan leer y escribir sin bloqueos manuales.
- **Newtonsoft.Json:** para generar el archivo metrics.json con los resultados de forma legible.
- **ScottPlot:** para crear gráficos de barras comparativos (tiempos, velocidad y eficiencia, procesadores).
- **YahooFinanceApi:** librería que facilita la consulta de precios de acciones en la nube.

Implementación Técnica

Descripción de la estructura del proyecto

El proyecto está organizado en clases bien definidas que separan claramente las responsabilidades. La clase SimulatedBrokerClient se encarga de simular la conexión con un broker, mostrando órdenes de compra o venta de acciones en un formato de tabla de consola. Por otro lado, la clase YahooFinanceService es la responsable de obtener los precios actualizados de las acciones utilizando la API de Yahoo Finance. Además, las operaciones de trading y consulta de precios están estructuradas de manera asíncrona para aprovechar mejor los recursos del sistema.

Explicación del código clave

Los fragmentos de código más importantes en nuestro programa incluyen:

- **Conexión y presentación de órdenes**: El método Connect inicializa la conexión simulada y muestra una cabecera en formato de tabla. Cada orden generada utiliza el método PlaceOrder, que da formato y presenta la orden con detalles como hora, tipo de transacción, cantidad, precio y agente responsable.

- **Obtención de precios**: La clase YahooFinanceService consulta los precios de las acciones de forma asíncrona. Se usa QueryAsync para solicitar únicamente los campos necesarios, haciendo el proceso más eficiente.

- **Paralelización**: Se destaca el uso de Parallel.ForEachAsync para ejecutar múltiples agentes de manera concurrente, distribuyendo la carga entre todos los núcleos disponibles del procesador.

Uso de mecanismos de sincronización

Aunque el proyecto no utiliza explícitamente mecanismos de sincronización tradicionales como locks o semaphores, se garantiza la correcta coordinación de tareas mediante programación asíncrona (async/await) y control de paralelismo con ParallelOptions. Esto evita bloqueos innecesarios y permite que múltiples operaciones se ejecuten de forma fluida y segura.

Justificación técnica de las decisiones tomadas

Cada decisión técnica fue pensada para lograr simplicidad, eficiencia y escalabilidad:

- **Uso de tareas asíncronas**: permite que el programa sea más rápido y responsivo, aprovechando mejor el tiempo de espera de operaciones externas como llamadas a la API.
- **Paralelización con control de concurrencia**: optimiza el uso del hardware disponible, reduciendo tiempos de ejecución de manera significativa.
- **Separación de responsabilidades**: tener clases específicas para el manejo de órdenes y obtención de precios facilita el mantenimiento y crecimiento futuro del proyecto.
- **Manejo de excepciones**: se incorporó control de errores para asegurar que el sistema pueda recuperarse o informar adecuadamente si ocurre algún problema durante la consulta de precios.

Evaluación de Desempeño

El simulador desarrollado permite comparar el rendimiento entre ejecuciones secuenciales y paralelas, proporcionando métricas clave que reflejan la eficiencia y escalabilidad de nuestro sistema.

Comparativa entre Ejecución Secuencial y Paralela

Se realizaron pruebas ejecutando 10 agentes, cada uno realizando 100 operaciones, tanto de forma secuencial como paralela. Estos resultados indican que la ejecución paralela mejora significativamente el rendimiento en comparación con la ejecución secuencial. Los resultados obtenidos fueron:

- **Tiempo de ejecución secuencial:** 1089 ms
- **Tiempo de ejecución paralela:** 233 ms
- **Speedup (aceleración):** 4.67x
- **Eficiencia:** 58.42%
- **Procesadores lógicos disponibles:** 8
- **Hilos utilizados:** 29
- **Operaciones totales:** 1000

Métricas: Tiempo de Ejecución, Eficiencia y Escalabilidad

- **Tiempo de Ejecución:** La ejecución paralela reduce el tiempo total de procesamiento, permitiendo completar las operaciones en un menor lapso.
- **Eficiencia:** Con una eficiencia del **58.42%,** se observa un buen aprovechamiento de los recursos disponibles, aunque existe margen para mejoras.
- **Escalabilidad:** El sistema muestra capacidad para escalar, ya que al aumentar el número de núcleos de CPU, se espera una mejora en el rendimiento, siempre que se mantenga un balance adecuado de carga y se minimicen las sobrecargas.

Gráficas o Tablas con Resultados

Los resultados de las métricas fueron exportados en formato gráfico y almacenados en la carpeta 'MetricsExport'. Estas visualizaciones permiten una interpretación más clara de los beneficios de la ejecución paralela sobre la secuencial.

Análisis de Cuellos de Botella o Limitaciones

A pesar de las mejoras observadas, se identificaron posibles cuellos de botella que podrían afectar el rendimiento. Identificar y mitigar estos cuellos de botella es esencial para mejorar la eficiencia y escalabilidad del sistema:

- **Sincronización de Acceso al Mercado:** El uso de mecanismos de sincronización, como locks, puede introducir latencias si múltiples agentes intentan acceder simultáneamente al recurso compartido.
- **Contención de Recursos:** La ejecución paralela puede generar competencia por recursos como CPU y memoria, especialmente si el número de hilos supera la capacidad del sistema.
- **Balance de Carga:** Es crucial distribuir equitativamente las tareas entre los agentes para evitar que algunos núcleos estén sobrecargados mientras otros están infrautilizados.

Trabajo en Equipo

- **3 Programadores**
  - Uno en agentes, otro en sincronización y otro en métricas
- **1 Coordinador:**
  - Planificación, documentación y presentaciones, comunicación diaria y control de versiones.

Herramientas utilizadas para coordinación

- **GitHub:** control de versiones, y revisiones de código colaborativas.
- **Excel:** elaboración de cronogramas, seguimiento de métricas de rendimiento y gestión de las tareas.
- **PDF:** distribución de la documentación técnica y de usuario en un formato accesible y estandarizado.
- **Microsoft Teams:** comunicación en tiempo real, reuniones diarias y coordinación de las actividades del equipo.

Conclusiones

Durante el desarrollo de la simulación, se aprendió a aprovechar la paralelización con Parallel.ForEachAsync, lo que mejoró significativamente el rendimiento respecto a la ejecución secuencial. También se destacó la importancia de sincronizar correctamente los datos compartidos entre agentes para evitar inconsistencias y mantener la integridad del sistema. El sistema mostró ser escalable, permitiendo aumentar la cantidad de agentes y aprovechar más núcleos sin necesidad de reestructurar el código.

Principales aprendizajes técnicos:

- Uso de la paralelización con Parallel.ForEachAsync para aprovechar al máximo los recursos del sistema.
- Importancia de sincronizar correctamente el acceso a datos compartidos para evitar inconsistencias entre agentes.
- Escalabilidad del sistema, permitiendo agregar más agentes o núcleos sin reestructurar el código.
- Mejora del rendimiento en comparación con la ejecución secuencial mediante el uso eficiente de los núcleos disponibles.

Retos enfrentados y superados:

- Encontrar una API que se acople a nuestro proyecto teniendo que probar con varias hasta encontrar una que funcione.
- Dificultades en la sincronización y manejo de acceso concurrente a los datos.
- Ajuste de la lógica de paralelización y control dinámico de hilos para mejorar el uso de los núcleos disponibles.

Posibles mejoras o líneas futuras:

- Optimización de la sincronización utilizando estructuras como ConcurrentQueue o SemaphoreSlim para mejorar la eficiencia del programa.
- Implementación de estrategias adaptativas que permitan seleccionar dinámicamente el enfoque de paralelización más adecuado.
- Expansión hacia la escalabilidad horizontal, permitiendo la ejecución de la simulación en varios servidores o contenedores.
- Incorporación de nuevas métricas de rendimiento, como la latencia de comunicación entre agentes, para identificar oportunidades de optimización más específicas.

Referencias

- https://learn.microsoft.com/es-es/dotnet/standard/parallel-programming/
- https://learn.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/threading/
- https://www.hanselman.com/blog/parallelforeachasync-in-net-6
- https://code-maze.com/csharp-parallel-foreachasync-and-task-run-with-when-all/
- https://es-us.finanzas.yahoo.com/
- https://www.sergigisbert.com/blog/programacion-paralela-en-c-con-la-clase-parallel/

Anexos

Capturas adicionales

Manual de uso adjuntado en **Github**.