Presentación
Grupo #5

Simulador de riesgo de inversión

Líder del grupo:
Luis Armando Martínez Méndez 2024-0202

Integrantes:

• Wilker José Capellán 2024-0217

• Emil Peralta Encarnación 2023-1151

• Nahuel Alexander Rodríguez 2023-1108

Carrera:
Desarrollo de Software

Profesor:
Erick Leonardo Pérez Veloz

Materia:
Programación Paralela

Fecha de Entrega:
12/12/2025

Indice

1. Introducción

❖ **Presentación general del proyecto**

Nuestro proyecto se trata de un simulador de riesgo financiero orientado a bancos, lo desarrollamos usando C# y Windows Forms para la interfaz, decidimos elegir este proyecto principalmente para dar respuesta para responder a una pregunta que sería: ¿qué tan arriesgada es una inversión y cuánto podemos esperar ganar?

Para hacer esta predicción, usamos una técnica avanzada llamada Simulación de Monte Carlo. En lugar de adivinar el futuro, este tipo genera miles de futuros posibles para un activo, basándose en datos económicos en su volatilidad histórica y la inflación.

Lo que hace especial a nuestra aplicación es que integra el paralelismo, es decir, en lugar de calcular los 50,000 futuros uno por uno, el programa los reparte y los calcula todos al mismo tiempo usando varios núcleos del procesador. Esto reduce el tiempo de espera de segundos a solo milisegundos, haciendo que el análisis de riesgo sea casi instantáneo.

❖ **Justificación del tema elegido**

Elegimos este tema principalmente porque es clase de programa que puede tener utilidad en el mundo real, por ejemplo, los bancos y fondos de inversiones desean tener una herramienta que les ayude a decidir sobre una inversión teniendo en cuenta el riesgo máximo de la misma y otros factores que intervienen en ese proceso. Para hacer un cálculo fiable, se necesitan miles de simulaciones para cubrir la mayoría de los futuros posibles, esto se enlaza con el otro motivo por el que decidimos realizar este proyecto, al tener que simulador miles de futuros, lo más eficiente es aplicar la programación paralela, así cubrimos el problema de las simulaciones, ya que se ejecutaran todas en paralelo y aparte demostramos la ventaja del paralelismo frente a la programación secuencial

❖ **Objetivos (general y específicos)**

A. Objetivo General

Construir una aplicación en C# que use el Paralelismo para ejecutar de manera velo la Simulación de Monte Carlo, permitiendo al usuario analizar el riesgo y el retorno de un activo financiero, mientras se mide y se demuestra la posible futura ganancia obtenida.

B. Objetivos Específicos

- **Modelado Preciso:** Implementar la fórmula del Movimiento Browniano Geométrico para simular trayectorias de precios de forma realista.
- **Aceleración del proceso:** Lograr que el tiempo de cálculo sea el mínimo posible utilizando la capacidad multihilo de la CPU, y documentar la mejora de rendimiento.
- **Experiencia de Usuario:** Crear una interfaz gráfica fácil de usar que reciba parámetros y muestre los resultados clave a través de gráficos informativos.
- **Toma de Decisiones:** Ofrecer métricas cruciales, como la Ganancia ajustada por Inflación y el Valor en Riesgo, para ayudar al usuario a determinar si la inversión es "Sólida" o "Especulativa".

2. Descripción del Problema

❖ **Contexto del problema seleccionado**

El problema central que aborda este proyecto es la incertidumbre inherente a la evolución de los precios de los activos financieros. En el mercado, los precios no siguen una trayectoria lineal o predecible; en cambio, se mueven de forma aleatoria, influenciados por factores externos (noticias, eventos económicos) y la propia volatilidad del activo.

Para gestionar esta incertidumbre, los analistas financieros necesitan cuantificar el riesgo potencial de pérdida. Esto se logra mediante la modelización matemática. El modelo elegido, el Movimiento Browniano Geométrico, asume que la variación del precio diario es aleatoria y sigue una distribución normal (la famosa "campana de Gauss").

El desafío es que, para obtener una estimación estadísticamente robusta (precisa) de dónde terminará el precio, no basta con simular una o diez posibles trayectorias, sino que se necesitan decenas de miles de escenarios. Este requisito de volumen es la fuente del problema computacional.

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

Cálculo del VaR (Valor en Riesgo): En un banco o fondo de inversión, es obligatorio calcular el VaR, que es la pérdida máxima probable que un portafolio puede experimentar en un horizonte de tiempo determinado con un nivel de confianza (ej., 95%). Una simulación de Monte Carlo permite estimar este VaR de manera precisa.

Decisión de Inversión: Un inversor necesita saber si el retorno esperado de un activo justifica el riesgo que está asumiendo. El simulador permite comparar el rendimiento promedio (la tendencia central) con la peor pérdida en el 5% de los escenarios (el VaR).

Mitigación de la Inflación: En un escenario económico real, la ganancia nominal debe ser ajustada por la tasa de inflación. El problema se agrava si el capital final, aunque positivo, tiene un valor real menor al capital inicial, resultando en una pérdida de poder adquisitivo. Nuestro simulador resuelve este problema al calcular la Ganancia Real.

- **Importancia del paralelismo en la solución**

El paralelismo es una parte clave de nuestra solución porque permite que el sistema trabaje más rápido y de manera más eficiente. En un análisis de riesgo financiero, necesitamos simular miles o incluso millones de posibles escenarios de inversión para poder estimar resultados, calcular probabilidades y evaluar qué tan riesgosa es una decisión. Hacer todo eso de forma secuencial tomaría demasiado tiempo.

Gracias al paralelismo, el sistema puede dividir el trabajo en varias partes usando la descomposición de datos**.** Esto significa que cada núcleo del procesador analiza una porción diferente de los datos al mismo tiempo. En lugar de esperar a que un solo núcleo haga todo, varios núcleos trabajan en equipo.

Este enfoque acelera la simulación y permite obtener resultados en menos tiempo, lo que es importante para tomar decisiones más rápidas y confiables. Además, el paralelismo mejora la precisión del análisis, ya que permite ejecutar más escenarios en el mismo período de tiempo, aumentando la calidad de la simulación sin afectar el rendimiento.

3. Cumplimiento de los Requisitos del Proyecto

1- Ejecución simultánea de múltiples tareas

El sistema implementa paralelismo de datos mediante la instrucción Parallel.For de la Task Parallel Library (TPL), lo que permite descomponer la carga de trabajo masiva (miles de simulaciones de Monte Carlo) en tareas concurrentes independientes. En lugar de una ejecución secuencial, el planificador de hilos de .NET distribuye dinámicamente las iteraciones entre todos los núcleos lógicos disponibles del procesador, logrando que múltiples trayectorias de precios futuros se calculen en el mismo instante exacto; esto maximiza la utilización del hardware y reduce drásticamente el tiempo de cómputo total.

2- Necesidad de compartir datos entre tareas

Para gestionar la escritura de resultados en una estructura común sin sacrificar rendimiento, se diseñó una estrategia de acceso a memoria libre de bloqueos (*lock-free*). Todos los hilos comparten la matriz resultados, pero la integridad de los datos se garantiza asignando a cada tarea un índice único de escritura, eliminando así las condiciones de carrera. Adicionalmente, se resolvió la complejidad de la generación de números aleatorios en paralelo instanciando objetos Random independientes con semillas únicas (Guid) dentro de cada hilo, asegurando independencia estadística y evitando los conflictos de concurrencia típicos de los generadores globales.

3- Exploración de diferentes estrategias de paralelización

Aunque la estrategia principal de nuestro proyecto es simple y eficiente, el diseño permite la experimentación práctica. Gracias a que podemos ajustar la cantidad de núcleos que la simulación debe usar, es posible medir y comparar cómo se comporta el rendimiento en diferentes configuraciones de hardware. Esto es clave para entender la eficiencia real del programa y demostrar cómo se aplica la Ley de Amdahl.

4- Escalabilidad con más recursos

El sistema es inherentemente escalable; si le añadimos más recursos de hardware, el tiempo de procesamiento disminuye casi de forma lineal. Esto se debe a que casi toda la lógica de cálculo es paralela, sin cuellos de botella significativos que dependan de un solo núcleo. La eficiencia del sistema se mide a través de la métrica de la eficiencia, la cual confirma que cada núcleo adicional está siendo aprovechado al máximo disminuir el tiempo de espera.

5- Métricas de evaluación del rendimiento

Para validar empíricamente la ventaja del paralelismo, el proyecto implementa métricas formales de computación de alto rendimiento y herramientas de experimentación controlada

Speedup: Medimos cuantitativamente cuánto más rápido es el sistema paralelo comparado con la ejecución secuencial.

Eficiencia: Evaluamos qué tan bien se están aprovechando los recursos de hardware adicionales.

Control manual del paralelismo: Se tiene libertad total del número de hilos que se quieren usar para la ejecución del programa.

Tiempo de cpu y Throughput: medicion precisa en milisegundos usando cronometros, medicion de cantidad de simulaciones procesadar por segundo para calculo bajo distintas cargas.

Convergencia de la Media: Confirmamos que, independientemente de la velocidad o el número de núcleos utilizados, la precisión matemática y los resultados financieros (VaR) se mantienen consistentes.

6- Aplicación a un problema del mundo real

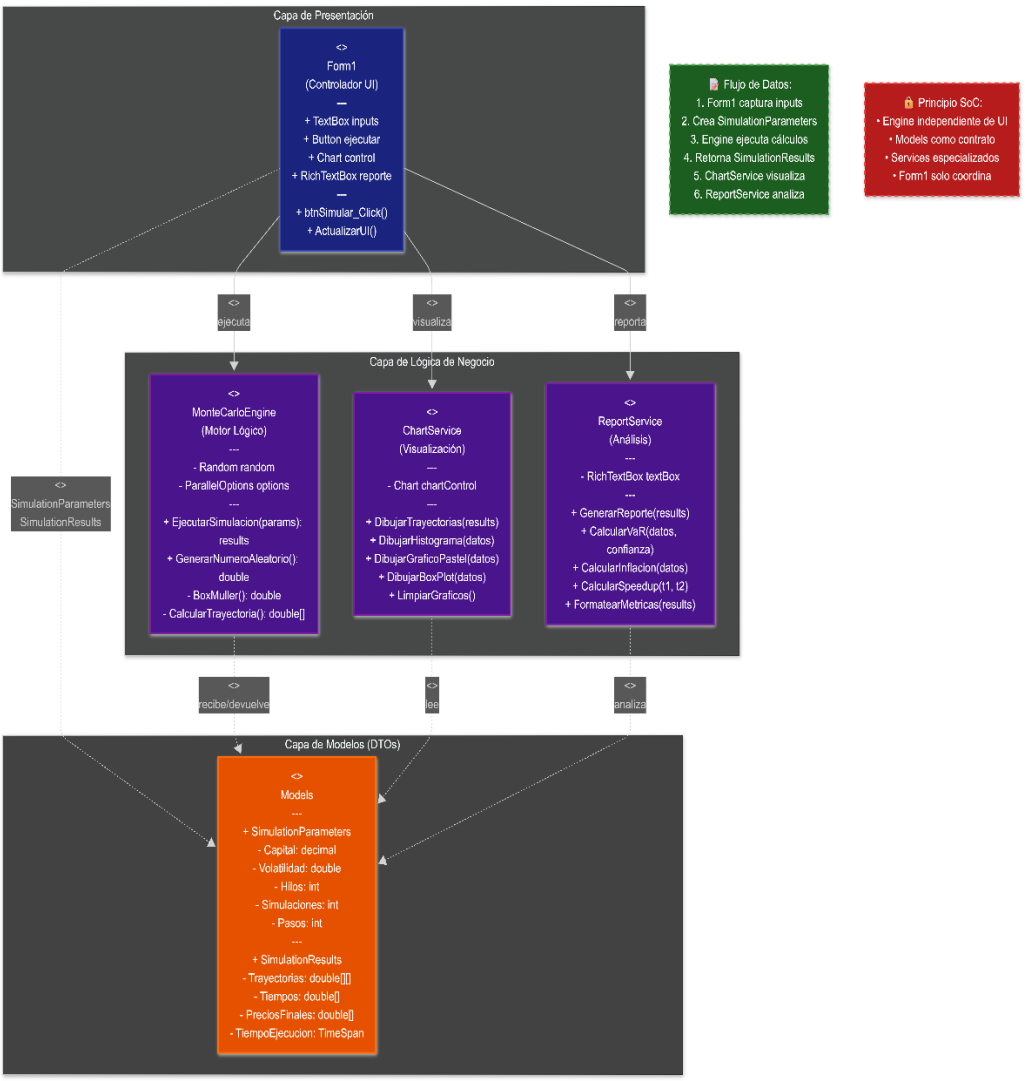
La aplicación básicamente aplica el poder de la computación paralela directamente a un problema crucial en finanzas que es la gestión de riesgos. El sistema implementa el modelo matemático que se llama Movimiento Browniano Geométrico para proyectar el futuro de un activo. Esto permite calcular métricas clave como el valor en riesgo, que es utilizado por los bancos para estimar la pérdida máxima en escenarios desfavorables y la ganancia real ajustando los resultados por el impacto de la inflación.

4. Diseño de la Solución

- **Arquitectura general del sistema**

La estructura que usamos sigue el patrón emisor y trabajadores. La UI viene a ser el Emisor, ya que se la encarga de recibir los datos del usuario y coordinar el trabajo. Mientras que los núcleos de la CPU vienen a ser los trabajadores, realizando la parte más complicada del cálculo que son las simulaciones. Una vez que terminan, el Emisor recopila y analiza los datos para generar el reporte.

❖ Diagrama de componentes/tareas paralelas



Este diagrama ilustra la estructura de Separación de Responsabilidades (SoC) implementada en el proyecto. El sistema se aleja de una arquitectura monolítica para adoptar un diseño por capas basado en servicios especializados.

Capa de Presentación (Form1):

Actúa exclusivamente como controlador y orquestador. Su única función es capturar los inputs del usuario, validar datos y coordinar las llamadas a los servicios inferiores.

Capa de Datos (Models):

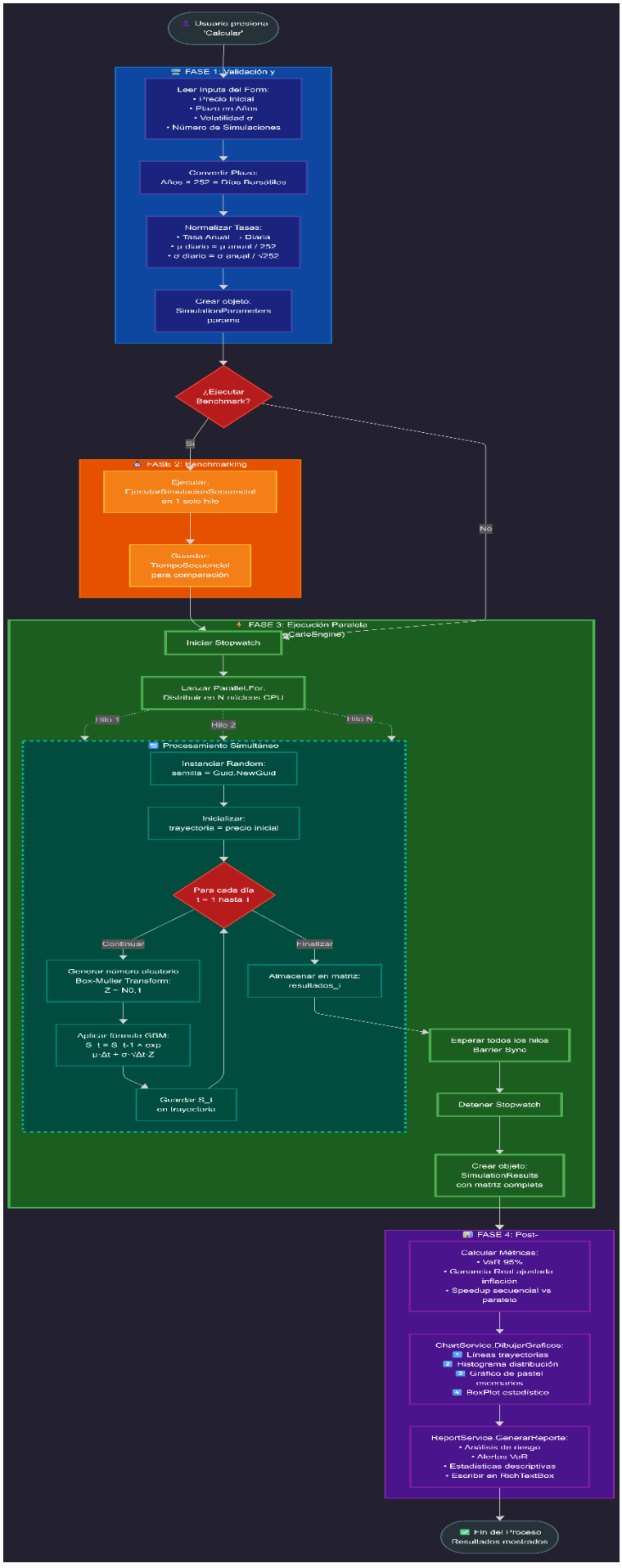
Define los Objetos de Transferencia de Datos (DTOs). SimulationParameters y SimulationResults actúan como el "contrato" o lenguaje común que permite que los componentes se comuniquen entre sí sin crear dependencias circulares.

Capa de Lógica de Negocio (Engine & Services):

MonteCarloEngine: Encapsula la complejidad algorítmica y el paralelismo. Es un componente puro que recibe parámetros y devuelve matrices numéricas.

-ChartService y ReportService: Segregan la lógica de visualización y análisis financiero, permitiendo que la forma de mostrar los datos evolucione independientemente del motor de cálculo.

❖ Diagrama de flujo del sistema



Este diagrama detalla la secuencia de operaciones desde que el usuario inicia la simulación hasta la presentación de resultados, haciendo énfasis en el procesamiento concurrente.

1. Validación y Preprocesamiento: El sistema normaliza las entradas (convirtiendo tasas anuales a diarias y plazos a días bursátiles) antes de iniciar cualquier cálculo.

2. Benchmarking (Fase Secuencial): Se ejecuta una instancia del algoritmo en un solo hilo (Main Thread) para establecer una línea base de rendimiento.

3. Núcleo Paralelo (Fase Crítica): Se utiliza Parallel.For para instanciar múltiples tareas simultáneas. El diagrama destaca lo que ocurre dentro de cada hilo:

-Aislamiento de Recursos: Cada hilo instancia su propio generador Random con semilla única para garantizar seguridad de hilos (thread-safety).

-Cálculo Estocástico: Se generan miles de trayectorias independientes aplicando la Transformada de Box-Muller y el modelo GBM.

4. Sincronización y Post-Proceso: Al finalizar todas las tareas paralelas (barrera de sincronización), el hilo principal consolida los resultados en memoria. Finalmente, los servicios de reporte y gráficos transforman estos datos crudos en información útil (VaR, Histogramas, Alertas de Riesgo) para el usuario final.

- **Estrategia de paralelización utilizada**

La estrategia es el Paralelismo de Datos, es utilizado para repartir el trabajo es el Particionamiento Dinámico, esto es para que los bloques de tareas se asignen a los núcleos sobre la marcha, cuando un núcleo termina su trabajo, se le asigna más inmediatamente, para que todos los procesadores estén ocupados de manera continua, esto lo que hace es optimizar el balanceo de carga. Además, la seguridad se garantiza porque se crean recursos independientes para cada hilo para evitar conflictos o algún bloqueo.

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

- Lenguaje y Framework: C# y .Net Core
- Procesamiento paralelo: Task.Parallel Library(TPL), Parallel.for y Plinq
- Interfaz gráfica: Windows Forms
- Métricas de tiempo: Stopwatch

5. Implementación Técnica

❖ Descripción de la estructura del proyecto

El sistema ha sido diseñado bajo una arquitectura modular basada en la Separación de Responsabilidades (SoC), descomponiendo la aplicación monolítica en cinco componentes especializados para facilitar la mantenibilidad y el trabajo colaborativo. El núcleo lógico reside en MonteCarloEngine, aislando completamente la complejidad matemática de la interfaz de usuario gestionada por Form1. La visualización y el análisis de datos se delegan a servicios independientes (ChartService y ReportService), permitiendo que la lógica de presentación evolucione sin afectar el motor de cálculo.

La comunicación entre estos módulos se realiza a través de un contrato de datos estricto definido en Models.cs.

Se utilizan las clases SimulationParameters para encapsular la entrada (capital, volatilidad, hilos, etc.) y SimulationResults para transportar la salida masiva de datos. Esta estructura garantiza que el flujo de información sea unidireccional y tipado, eliminando dependencias circulares y permitiendo que cada integrante del equipo trabaje en una capa específica sin generar conflictos de integración.

❖ Explicación del código clave

El componente central del sistema es el método EjecutarSimulacion dentro de la clase MonteCarloEngine.

Este método implementa el algoritmo de Movimiento Browniano Geométrico (GBM) utilizando la instrucción Parallel.For de la Task Parallel Library (TPL).

A diferencia de un bucle convencional, esta instrucción particiona el espacio de iteraciones (simulaciones) y las distribuye dinámicamente entre los núcleos lógicos disponibles, permitiendo al usuario controlar el grado de paralelismo mediante la clase ParallelOptions.

Dentro de la ejecución paralela, se aplica la transformación matemática de Box-Muller para generar números aleatorios con distribución normal estándar a partir de una fuente uniforme.

Esto es crucial para simular la volatilidad financiera de manera realista. Adicionalmente, se implementó un método espejo EjecutarSimulacionSecuencial que replica la carga matemática en un solo hilo, sirviendo como línea base (benchmark) para calcular métricas precisas de aceleración (Speedup) y eficiencia en el reporte final.

❖ Uso de mecanismos de sincronización

Se adoptó una estrategia de concurrencia sin bloqueos (lock-free). Se pre-asigna una matriz de memoria compartida (double) donde cada hilo escribe exclusivamente en un índice único (i) garantizado por el bucle paralelo. Esto elimina por diseño las condiciones de carrera (Race Conditions) en la escritura de datos.

Respecto a la generación de números aleatorios, se resolvió el problema de seguridad de hilos (thread-safety) inherente a la clase Random de .NET evitando una instancia global compartida. En su lugar, se instancia un objeto Random local dentro de cada tarea paralela, inicializado con una semilla única basada en Guid.NewGuid().GetHashCode(). Esto asegura la independencia estadística de cada trayectoria simulada y evita la contención de recursos que ocurriría si múltiples hilos intentaran acceder al mismo generador simultáneamente.

❖ Justificación técnica de las decisiones tomadas

La decisión de implementar una arquitectura de Servicios Desacoplados (ChartService, ReportService) en lugar de incluir la lógica en el formulario principal responde a la necesidad de testabilidad y escalabilidad. Al separar el cálculo matemático de la representación visual, es posible ejecutar pruebas unitarias sobre el motor de Monte Carlo o cambiar la biblioteca de gráficos (ej. a una web o móvil) sin reescribir el núcleo del algoritmo. Esto demuestra un diseño orientado a largo plazo, superior a una implementación ad-hoc.

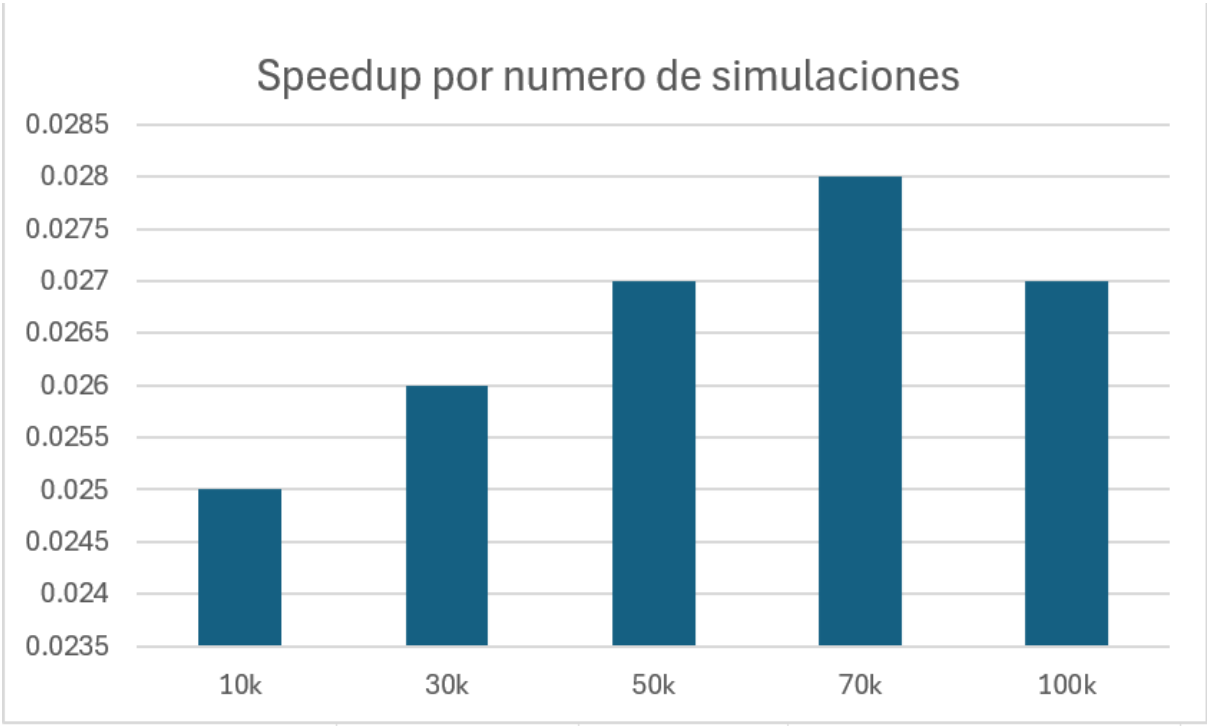
Adicionalmente, se optó por el uso de Arreglos Primitivos (double y double) en lugar de colecciones genéricas como List<T> para el almacenamiento de las trayectorias masivas. Esta decisión técnica se fundamenta en la eficiencia de memoria y el rendimiento de la caché de la CPU.

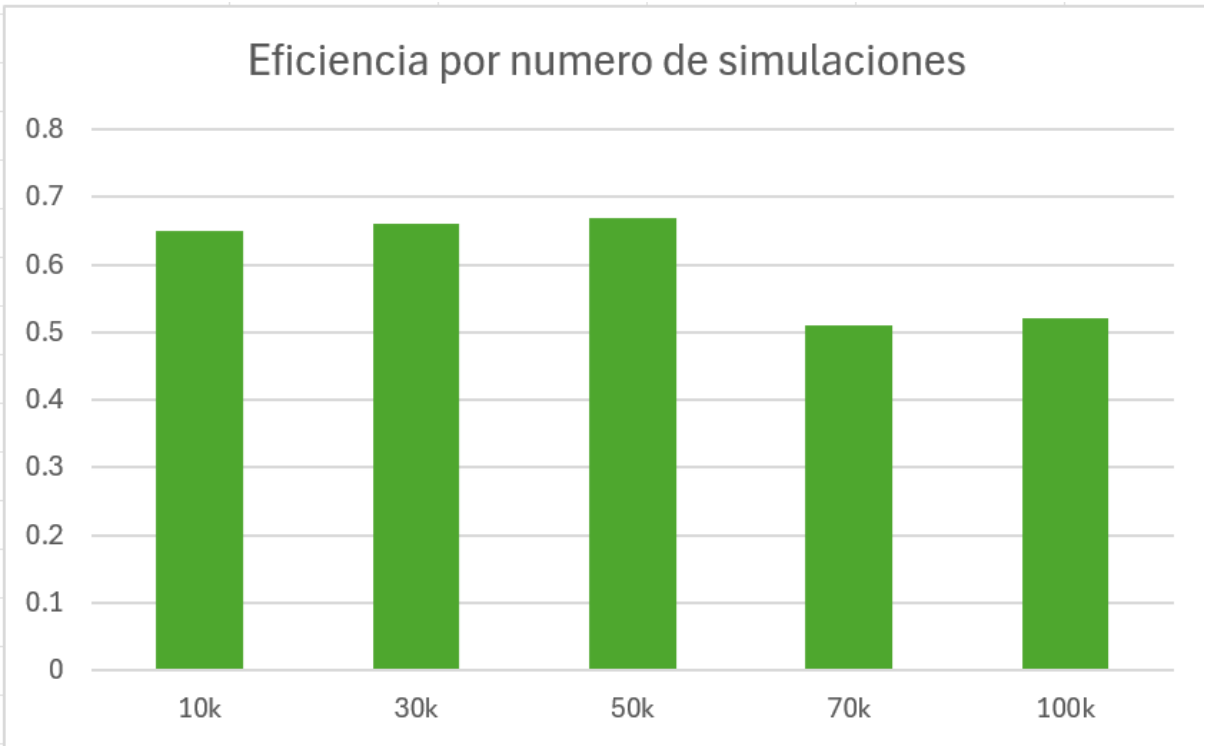
Los arreglos garantizan bloques de memoria contiguos, lo que optimiza el prefetching del procesador y reduce drásticamente la sobrecarga del Garbage Collector (GC) durante simulaciones de alta frecuencia, resultando en una ejecución significativamente más rápida y estable.

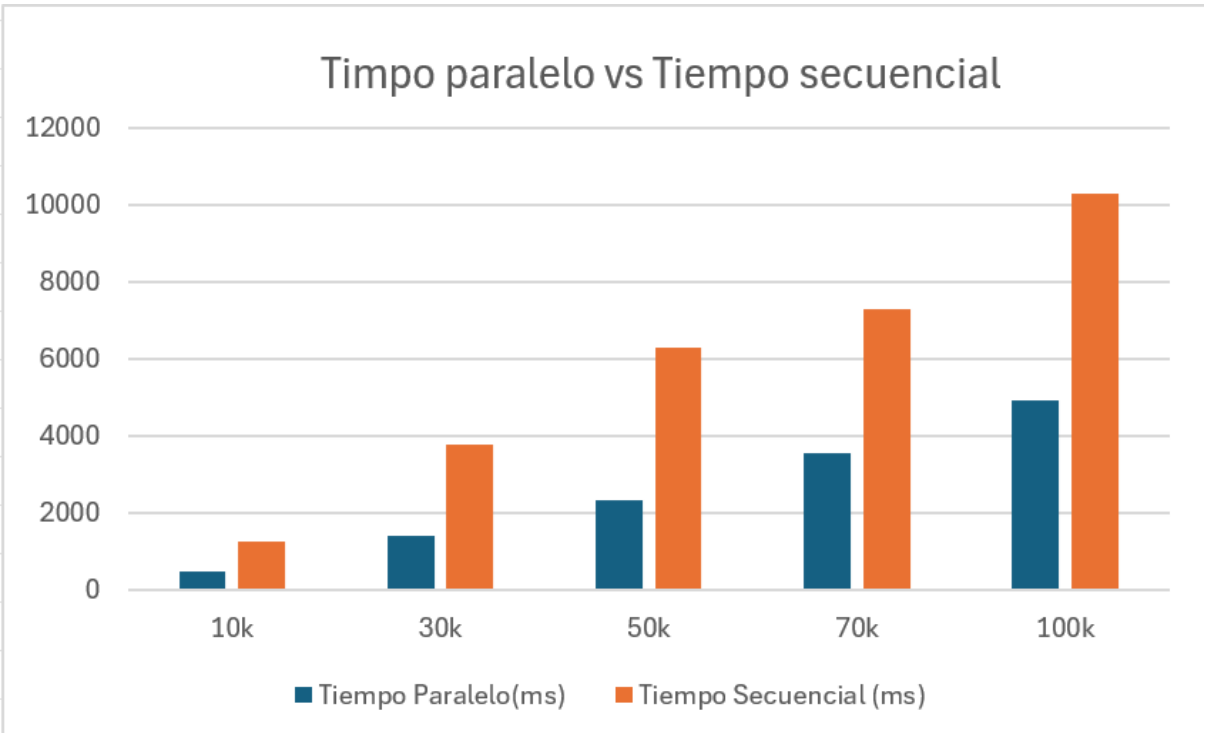
6. Evaluación de Desempeño

Para evaluaciones de desempeño utilizamos diferentes números de simulaciones de 10 mil simulación hasta 100 mil, para evaluar el rendimiento del programa.

Numero de simulacones	Speed up	Eficiencia	Tiempo Paralelo(ms)	Tiempo Secuencial (ms)
10, 000	0.025	0.65	481	1255
30, 000	0.026	0.66	1415	3787
50, 000	0.027	0.67	2330	6301
70, 000	0.028	0.51	3544	7299
100,000	0.027	0.52	4919	10287




Eficiencia por numero de simulaciones
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10k
30k
50k
70k
100k


Timpo paralelo vs Tiempo secuencial
12000
10000
8000
6000
4000
2000
0
10k
30k
50k
70k
100k
Tiempo Paralelo(ms)
Tiempo Secuencial (ms)

❖ **Análisis de cuello de botella o limitaciones**

1. Cuello de botella en el gráfico

Aunque el sistema usa paralelismo y calcula muy rápido, todavía tiene limitaciones importantes. La primera es el rendimiento del gráfico: aunque los cálculos se hacen en milisegundos, mostrar miles de trayectorias en pantalla hace que la interfaz se ponga lenta o incluso se congele. Esto ocurre porque el control gráfico de Windows Forms no está optimizado para manejar tantos datos a la vez.

2. Alto consumo de memoria

Otra limitación es el consumo de memoria. El sistema guarda todos los datos de cada simulación en la RAM. Si el usuario coloca cantidades muy grandes de simulaciones, la memoria disponible podría no ser suficiente y la aplicación terminaría dando un error por falta de memoria.

3. Errores de precisión

También existe un pequeño margen de error debido al tipo de dato utilizado (double). Este tipo es muy rápido, pero introduce micro-errores de redondeo. Para una simulación de riesgo esto no es un problema, pero no sería adecuado para cálculos financieros exactos que requieren precisión al centavo.

4. Limitación del paralelismo

El sistema además solo aprovecha los núcleos de una sola computadora. No puede distribuir las simulaciones entre varias máquinas ni usar aceleración por GPU, lo que limita su escalabilidad cuando se intenta procesar cantidades extremadamente grandes de datos.

5. Limitaciones del generador aleatorio

Por último, el generador de números aleatorios que se usa es el estándar de .NET. Funciona bien para simulaciones comunes, pero no para estudios científicos o de seguridad que requieren un nivel de aleatoriedad mucho más alto

7. Trabajo en Equipo

El proyecto fue Elaborados en conjunto entre todos los integrantes del equipo, donde a cada integrante de este le fue asignada una serie de responsabilidades esenciales para así favorecer el avance del proyecto. Para esto fue usada la herramienta de Microsoft Exel.

Integrante	Responsabilidades en el proyecto	Descripcion
Wilker Jose Capellan	Responsable de implementar el motor paralelo para las simulaciones del programa del programa Responsable de los puntos 3 y 4 de la documentación del proyecto	Fue responsable de la lógica de cálculo de Monte Carlo. Implementa tanto la versión secuencial como la paralela, y se encarga de la descomposición de datos para la ejecución en multihilo.
Emil Peralta	Encargado del análisis y procesamiento de resultados Responsable de los puntos 6 y 9 de la documentación del proyecto	Es el responsable de recibir los datos finales de la simulación. Su tarea es calcular las métricas clave (Ganancia Real, Speedup, Eficiencia) y generar todo el reporte de texto final con el análisis de decisión para la interfaz.
Nahuel Molina	Encargado de la creación de los gráficos Responsable de los puntos 5 y 10 de la documentación del proyecto	Fue responsable de toda la sección visual de resultados. Implementa la clase base para los gráficos y crea las funciones específicas para generar los diferentes tipos de gráficos , como Líneas de trayectorias, Histograma, Pastel, Barras y Bigotes.
Luis Armando Martinez	Encargado del programa principal, la interfaz grafica y la coordinacion del equipo en desarrollo del proyecto Responsable de los puntos 1, 2 y 7 de la documentación del proyecto	Es el responsable de crear toda la Interfaz Gráfica, crear todos los controles, botones, campos de texto, etc y conectar conectar todos los módulos, aparte fue el encargado de la coordinación del equipo en el desarrollo del proyecto.

8. Conclusiones

Retos Principales aprendizajes técnicos

- **Luis Armando Martinez Mendez:** Trabajé en la construcción de la interfaz gráfica del sistema y el ordenamiento de los módulos del programador, más aparte como líder estuve coordinando al equipó en la construcción, estos conocimiento me ayudaron a desarrollar mis habilidades como programador y como líder.

- **Wilker Jose Capellan:** Desarrollar este simulador enriqueció mucho mis conocimientos de programación, aprendiendo mucho sobre la eficiencia en el procesamiento masivo de información y que no podemos depender solo de la lógica, también hay que dominar las herramientas de paralelismo y gestión de memoria.

- **Nahuel Molina:** Yo aprendi cosas basicas hasta dentro de Github y entendi cómo organizar correctamente un proyecto usando ramas, commits bien estructurados y pull requests y también pude aprender a localizar problemas en el codigo y también aprendí un poco sobre como trabajar entre Visual Studio y GitHub.

- **Emil Peralta:** Lo que aprendi fue que es mejor hacer una capa de salida totalmente separada para que no haya ningun problema para imprimir el reporte y tambien aprendí a como trabajar en un quipo con roles definidos y organizados.

❖ **Retos enfrentados y superados**

- **Luis Armando Martinez Mendez:** Uno de los principales desafíos fue realizar la construcción de interfaz en sí, tambien la entrada de datos y conectarlos con los módulos para que el programa pueda funcionar. Otro reto fue lograr un balance estético lo siguientemente y funcionalidad, para esto use windows forms chart para construirla. Por último, el reto fue aprender a coordinar un equipo, nunca había trabajado de esta forma y considero que los resultados fueron satisfactorios.

- **Wilker Jose Capellan:** El mayor desafio fue cambiar la mentalidad secuencial para el diseño de la arquitectura lock-free y solucionar los problemas de concurrencia al manipular los hilos de forma segura.

- **Nahuel Molina:** El principal reto fue corregir el código del servicio de gráficos, porque primero no terminaba de encajar la idea y después también me costó entender el orden correcto para trabajar con las ramas y cómo dividir los cambios en varios commits, y los métodos fuera de clases pero con paciencia y preguntando lo resolvi.

- **Emil Peralta:** Mi reto fue calcular y presentar los resultados como el Speedup y la Eficiencia a partir de los tiempos de ejecución, también la ganancia real y nominal. Con esto entendí cómo usar el Speedup correctamente y la Eficiencia para probar que el código paralelo es realmente más rápido, tambien entendi como optimizar el flujo de datos entre el motor y el reporte

9. Fuentes

Microsoft Docs – Parallel Programming in .NET

- https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/

Microsoft Docs – Task Parallel Library (TPL)

- https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl

Microsoft Docs – Parallel.For and Parallel.ForEach in C#

- https://learn.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel

Glasserman, P. (2004). Monte Carlo methods in financial engineering. Springer.

- https://link.springer.com/book/10.1007/978-0-387-21617-1

10. Anexos

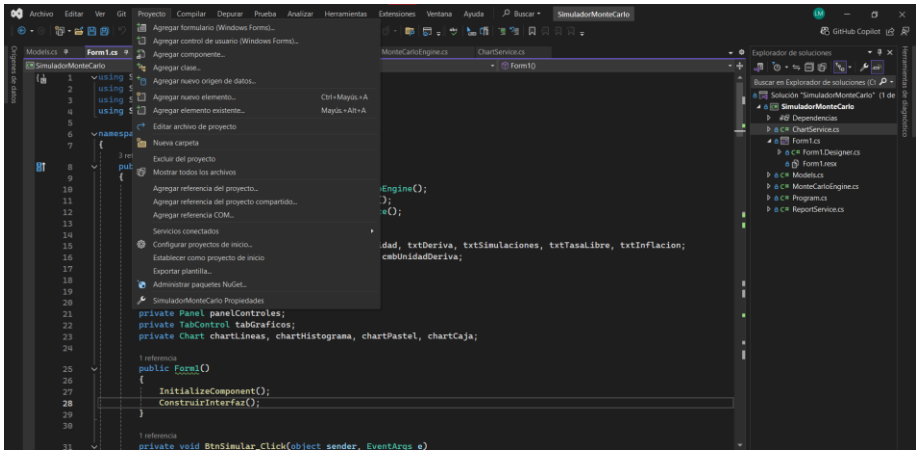
❖ Manual de ejecución del sistema

Pre-requisitos:

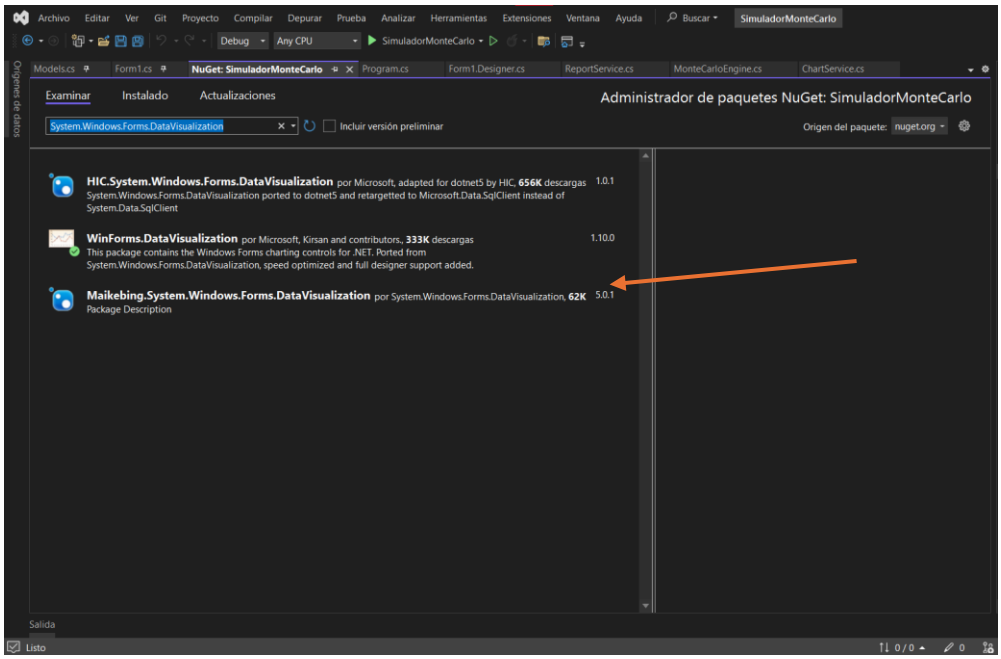
1. Tener instalado .Net instalado en el equipo y asegurarse de tener la plantilla de windows forms.

2. Tener instalado el paquete de librerías Windows.Forms.DataVizualation en el caso de que no lo tenga siga los siguientes pasos:

1. Dirígete al administrador de paquetes Nuggets



2. Busca System.Windows.Forms.DataVizualation



Pasos para ejecutar el programa

1. Ir al link del repositorio y clonarlo a la ultima version:

https://github.com/Luis-Armando-Martinez-Mendez/Proyecto-final-Grupo-2.git

2. Entrar a VisualStudio o Visual Studio Code y entrar a la carpeta del proyecto

3. Darle a el botón de run asegurando que se ejecute el form1 como principal

Al ejecutarse el programa permite:

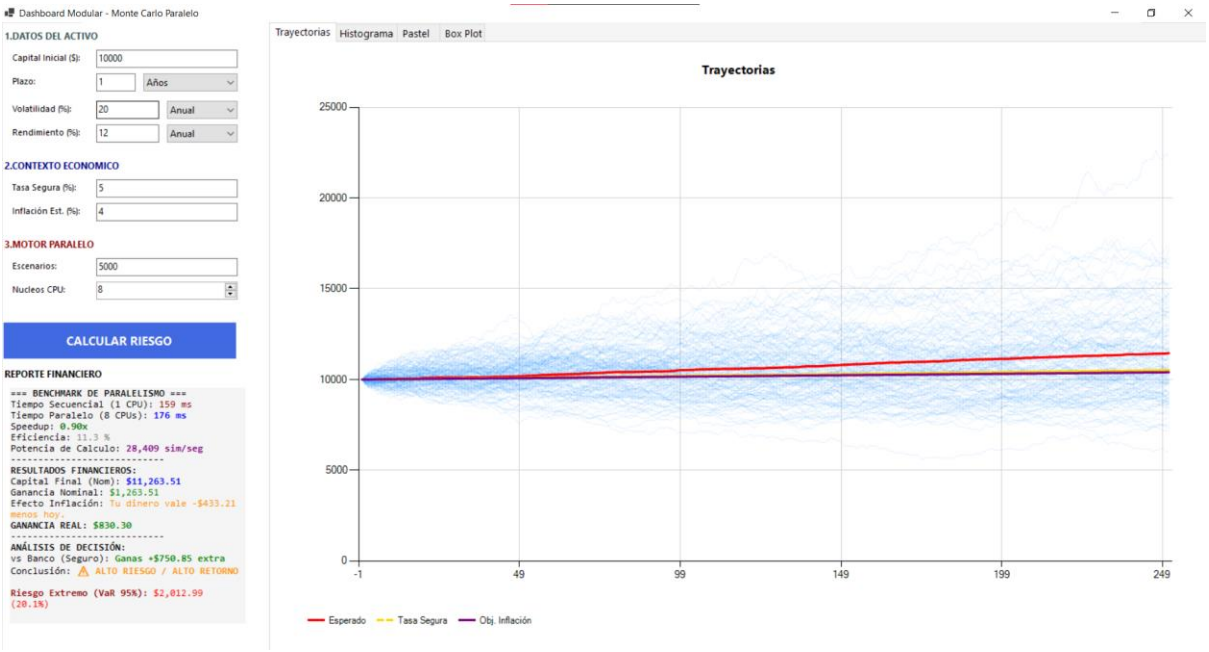
Ingresar los valores para la simulación:

- Capital inicial
- Tiempo (años, meses o dias)
- Rendimiento
- Volatilidad
- Inflación
- Número de escenarios
- Nucleos a utilizar

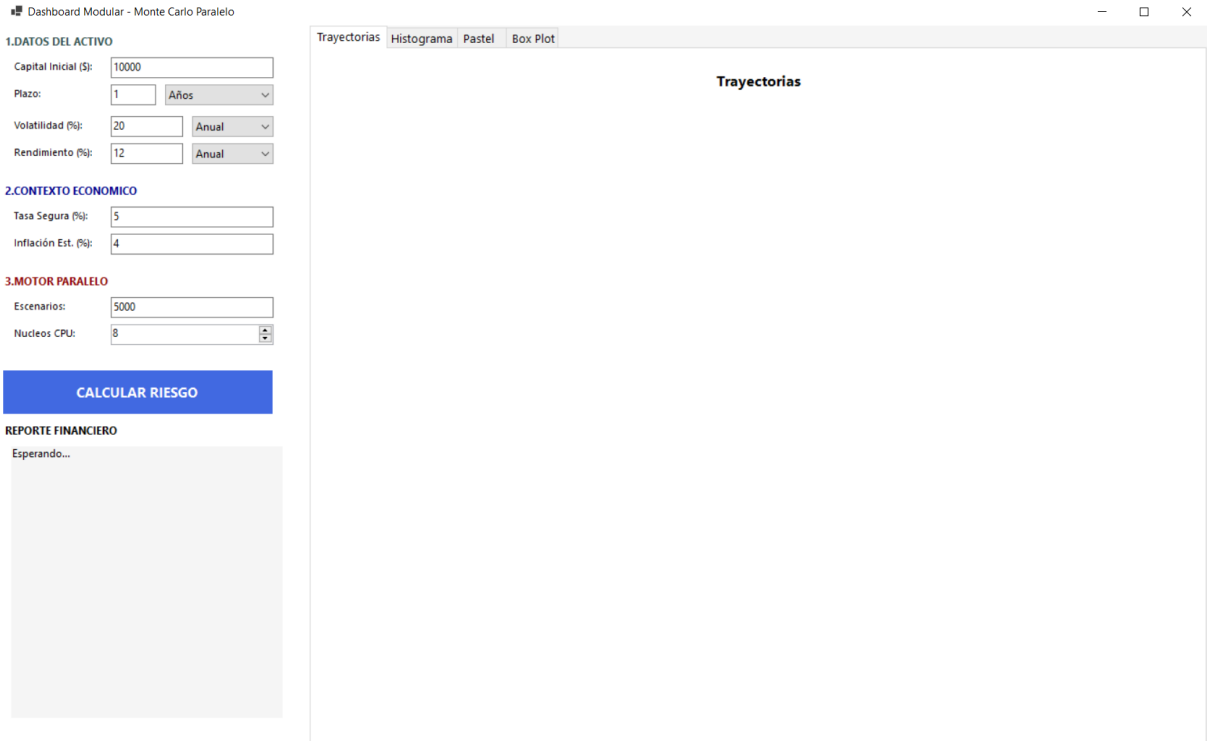
El programa calcula:

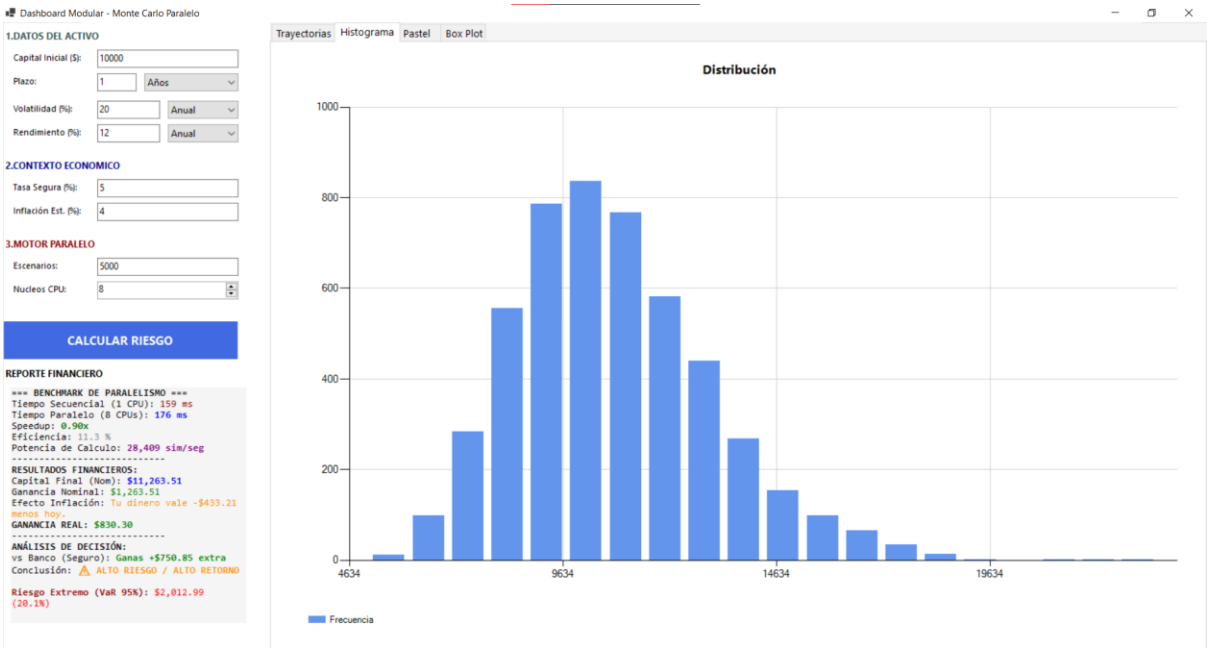
- Tiempo secuencial
- Tiempo paralelo
- Speed up
- Eficiencia
- Posiblemente ganacia

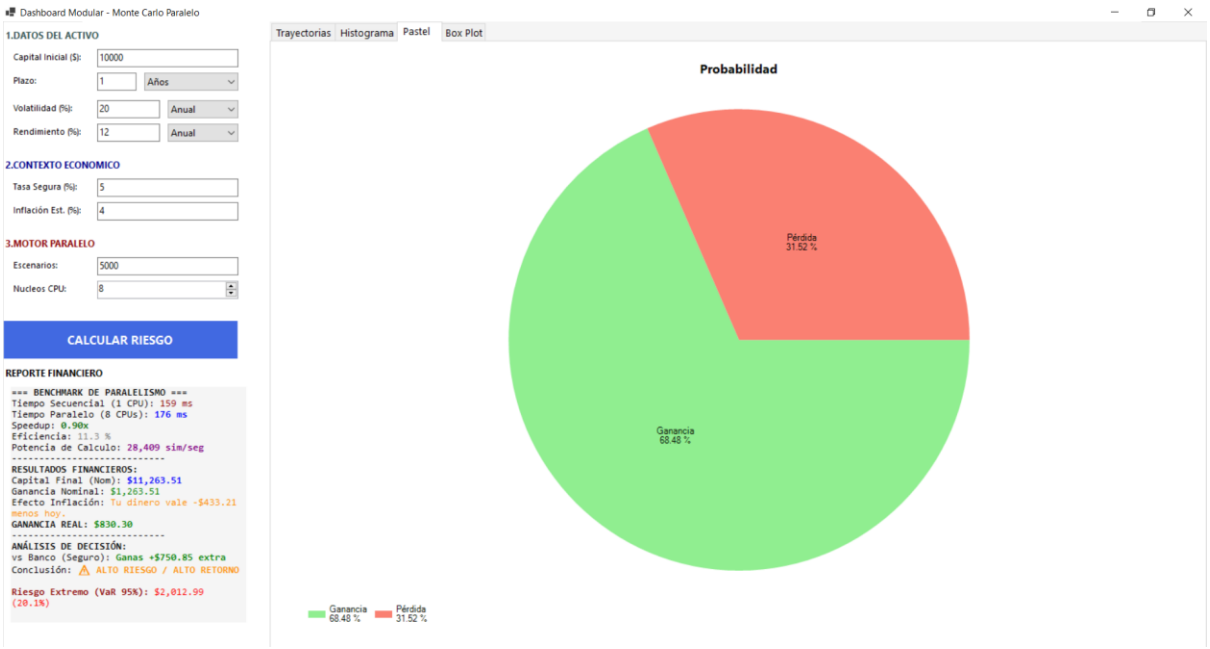
El resultado del seguimiento de los pasos se debería ver así:

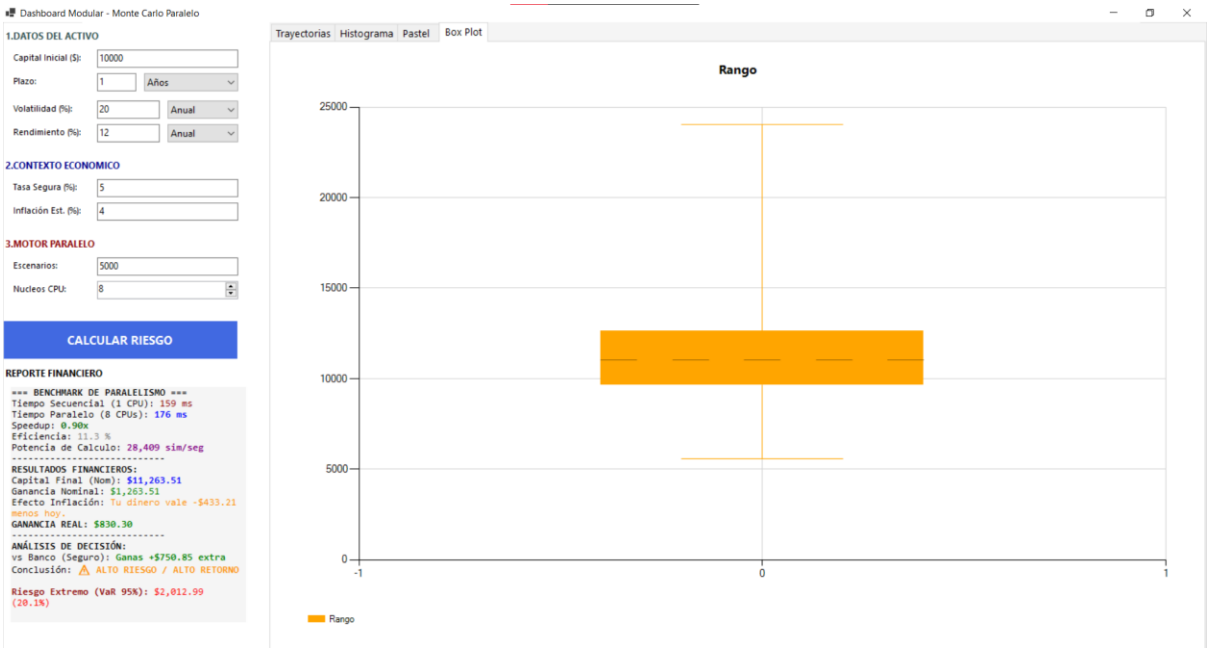


❖ Capturas adicionales




Dashboard Modular - Monte Carlo Paralelo
1.DATOS DEL ACTIVO
Capital Inicial ($): 10000
Plazo: 1 Años
Volatilidad (%): 20 Anual
Rendimiento (%): 12 Anual
2.CONTEXTO ECONOMICO
Tasa Segura (%): 5
Inflación Est. (%): 4
3.MOTOR PARALELO
Escenarios: 5000
Nucleos CPU: 8
CALCULAR RIESGO
REPORTE FINANCIERO
=== BENCHMARK DE PARALELISMO ===
Tiempo Secuencial (1 CPU): 159 ms
Tiempo Paralelo (8 CPUs): 176 ms
Speedup: 0.90x
Eficiencia: 11.3 %
Potencia de Calculo: 28,409 sim/seg
RESULTADOS FINANCIEROS:
Capital Final (Nom): $11,263.51
Ganancia Nominal: $1,263.51
Efecto Inflación: Tu dinero vale -$433.21 menos hoy.
GANANCIA REAL: $830.30
ANÁLISIS DE DECISIÓN:
vs Banco (Seguro): Ganas +$750.85 extra
Conclusión: ALTO RIESGO / ALTO RETORNO
Riesgo Extremo (VaR 95%): $2,012.99 (20.1%)
Trayectorias Histograma Pastel Box Plot
Distribución
1000
800
600
400
200
0
4634
9634
14634
19634
Frecuencia


Dashboard Modular - Monte Carlo Paralelo
1.DATOS DEL ACTIVO
Capital Inicial ($): 10000
Plazo: 1 Años
Volatilidad (%): 20 Anual
Rendimiento (%): 12 Anual
2.CONTEXTO ECONOMICO
Tasa Segura (%): 5
Inflación Est. (%): 4
3.MOTOR PARALELO
Escenarios: 5000
Nucleos CPU: 8
CALCULAR RIESGO
REPORTE FINANCIERO
=== BENCHMARK DE PARALELISMO ===
Tiempo Secuencial (1 CPU): 159 ms
Tiempo Paralelo (8 CPUs): 176 ms
Speedup: 0.90x
Eficiencia: 11.3 %
Potencia de Calculo: 28,409 sim/seg
RESULTADOS FINANCIEROS:
Capital Final (Nom): $11,263.51
Ganancia Nominal: $1,263.51
Efecto Inflación: Tu dinero vale -$433.21 menos hoy.
GANANCIA REAL: $830.30
ANÁLISIS DE DECISIÓN:
vs Banco (Seguro): Ganas +$750.85 extra
Conclusión: ALTO RIESGO / ALTO RETORNO
Riesgo Extremo (VaR 95%): $2,012.99 (20.1%)
Trayectorias Histograma Pastel Box Plot
Probabilidad
Pérdida
31.52 %
Ganancia
68.48 %
Ganancia
68.48 %
Pérdida
31.52 %



❖ Enlace al repositorio de Git

https://github.com/Luis-Armando-Martinez-Mendez/Proyecto-final-Grupo-2.git