



**Nombre del curso:**

Programación Paralela

**Título del proyecto:**

Sistema De inventario Especulativo

**Integrantes del equipo:**

Berlyng Manuel Yena Garcia 2023-1890

Carlos Manuel Reynoso Ramón 2023-1102

Elys Camila Batista Encarnación 2023-1111

**Líder:**

Wilmar Sanchez Suarez 2024-0200

**Maestro:**

Erick Leonardo Pérez

**Fecha de entrega:**

12/12/25

## **INDICE**

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 .....	10
7. Trabajo en Equipo.....	11
8. Conclusiones.....	12
9. Referencias.....	15
10. Anexos.....	15

# 1. Introducción

## Presentación general del proyecto

El proyecto que hemos desarrollado se llama **Sistema de Inventario Especulativo**. Consiste en una aplicación de consola diseñada para simular y analizar el comportamiento de un inventario masivo de productos en un escenario de venta. El sistema no solo almacena datos, sino que procesa millones de registros de productos, calculando predicciones de demanda futura (si la venta subirá, bajará o se mantendrá estable) basándose en el historial de ventas recientes y la categoría del producto. La característica central del proyecto es la implementación de dos modos de ejecución: uno secuencial y otro paralelo, lo que nos permite medir y comparar directamente el rendimiento del hardware moderno bajo carga.

## Justificación del tema elegido

Elegimos desarrollar un sistema de predicción de inventario porque representa un problema muy real en la industria, el manejo de Big Data. En el mundo real, las empresas no analizan diez o veinte productos, sino millones, y necesitan respuestas rápidas para tomar decisiones de reabastecimiento u ofertas. La justificación técnica es demostrar que la programación secuencial tiene un límite cuando el volumen de datos crece. Queríamos aplicar los conceptos de Programación Paralela para probar cómo, al dividir el trabajo entre los múltiples núcleos del procesador, podemos reducir los tiempos de espera. Este proyecto es la prueba práctica de que utilizar correctamente los recursos del hardware impacta directamente en la eficiencia del software.

## Objetivos (general y específicos)

### Objetivos

**Objetivo General** Desarrollar un sistema de software en C# que procese un gran volumen de datos de inventario utilizando técnicas de paralelismo, con el fin de comparar su rendimiento contra una ejecución secuencial y verificar la ganancia en velocidad Speedup y eficiencia.

### Objetivos Específicos

1. **Generar datos masivos:** Crear un módulo capaz de simular una base de datos de millones de productos con atributos aleatorios como categoría, stock y ventas recientes para estresar el sistema.
2. **Implementar lógica:** Desarrollar un Motor Especulativo que aplique reglas matemáticas y factores de probabilidad sesgos por categoría como Alimentos o Electrónica para predecir la demanda real.

3. **Paralelizar el procesamiento:** Utilizar la librería System.Threading.Tasks y el bucle Parallel.For para distribuir la carga de análisis entre los núcleos disponibles del procesador.
4. **Medir y comparar métricas:** Implementar contadores de tiempo Stopwatch para calcular exactamente cuántos milisegundos toma cada enfoque y derivar el Speedup y la Eficiencia final de la solución.

## 2. Descripción del Problema

### Contexto del problema seleccionado

El problema central que abordamos es el cuello de botella en el procesamiento de datos. En la programación tradicional (secuencial), las instrucciones se ejecutan una detrás de otra. Esto funciona bien si tienes pocos datos, pero nuestro proyecto simula un entorno con millones de registros. Al intentar analizar tal cantidad de información línea por línea para predecir tendencias de mercado, el procesador se ve obligado a trabajar con un solo núcleo mientras los demás están inactivos. Esto genera tiempos de espera inaceptables para una aplicación que debería ser ágil. El desafío técnico no es solo hacer que funcione, sino hacer que aproveche todo el hardware disponible.

### Aplicación del problema en un escenario real

Imaginemos una gran cadena de supermercados o un gigante del comercio electrónico tipo Amazon o Mercado Libre que necesita actualizar sus precios o stock en tiempo real. Nuestro sistema utiliza categorías como Alimentos, Electrónica, Limpieza y Hogar. En la vida real, un gerente necesita saber ya mismo si la demanda de televisores va a bajar (para lanzar una oferta) o si la demanda de alimentos va a subir (para reabastecer). Si el sistema tarda horas en calcular esto, la oportunidad de negocio se pierde. Por ejemplo, nuestro algoritmo simula que la Electrónica tiende a bajar de precio y demanda rápidamente; detectar esto a tiempo en millones de productos permite a la empresa no quedarse con inventario.

### Importancia del paralelismo en la solución

Aquí es donde entra la Programación Paralela. La importancia de aplicar paralelismo radica en transformar un proceso lento en uno rápido sin cambiar la lógica del negocio, solo la forma en que se ejecuta. Al usar herramientas como Parallel.For, dividimos esos millones de productos en pequeños bloques que se procesan simultáneamente en todos los núcleos de la CPU. Esto es muy importante porque:

1. **Reduce el tiempo de respuesta:** Pasamos a reducir en torno a la mitad el tiempo esperado

2. **Optimiza recursos:** Usamos el 100% de la capacidad de la máquina en lugar del 10% o 20% que es común en procesos secuenciales.
3. **Escalabilidad:** Si la empresa crece a 10 millones de productos, la solución paralela aguantará mucho mejor la carga que la secuencial.

### 3. Cumplimiento de los Requisitos del Proyecto

**Para cada criterio, se debe justificar cómo el proyecto lo aborda:**

#### 1. Ejecución simultánea de múltiples tareas

El núcleo de nuestro proyecto no procesa los productos uno por uno. Gracias a la implementación del bucle Parallel.For, logramos que el análisis de los productos se distribuya entre los diferentes núcleos del procesador. Mientras un hilo está calculando la predicción para el Producto 100 evaluando sus tendencias de subida o bajada, otro hilo distinto está haciendo exactamente lo mismo para el Producto 101 al mismo tiempo. No hay espera todo ocurre en simultáneo aprovechando el hardware al máximo.

#### 2. Necesidad de compartir datos entre tareas

Todas las tareas paralelas necesitan acceder al inventario. En nuestro sistema, generamos una lista masiva de objetos Producto al inicio que contiene el ID, categoría, stock y ventas históricas. Este dataset está en memoria y es compartido por todos los hilos. El reto aquí fue asegurar que múltiples hilos pudieran leer las ventas históricas de diferentes productos al mismo tiempo sin causar conflictos ni bloqueos, permitiendo que el MotorEspeculativo tome los datos que necesita para sus cálculos sin duplicar información innecesariamente.

#### 3. Exploración de diferentes estrategias de paralelización

El diseño es flexible. Aunque ahora procesamos 2 millones de registros, la lógica del Parallel.For permite que el sistema escala si el hardware mejora.

#### 4. Escalabilidad con más recursos

- **Con más recursos:** Si ejecutamos este programa en un servidor con 32 núcleos, el sistema dividirá el trabajo en más trozos automáticamente, reduciendo el tiempo total sin que tengamos que cambiar una sola línea de código.
- **Con más datos:** Si subimos la carga a 10 millones de productos, el sistema simplemente usará más ciclos de CPU, pero la estructura paralela evitará que el programa se "congele", cosa que sí pasaría en la versión secuencial.

#### 5. Métricas de evaluación del rendimiento

En el Program.cs implementamos un sistema de medición usando la clase Stopwatch para capturar:

**Tiempo de Ejecución:** Cuántos milisegundos tarda la versión secuencial vs. la paralela.

**Speedup:** Calculamos cuántas veces más rápido es el paralelo dividiendo el tiempo secuencial entre el paralelo.

**Eficiencia:** Evaluamos qué tan bien usamos los recursos dividiendo el Speedup entre la cantidad de procesadores lógicos. Si nos acercamos a 1.0, significa que no estamos desperdiando potencia.

## 6. Aplicación a un problema del mundo real

Este proyecto simula un escenario crítico en la logística moderna. Empresas como supermercados, Amazon o grandes distribuidoras no pueden permitirse revisar sus inventarios manualmente. Nuestro Motor Especulativo replica la lógica que usan los departamentos de compras: analizar el historial de ventas y predecir si un producto se va a agotar para pedir más o si se va a estancar para ponerlo en oferta. Resolver esto en segundos y no en horas es lo que diferencia a una empresa eficiente de una que pierde dinero, y por eso el paralelismo es la solución adecuada.

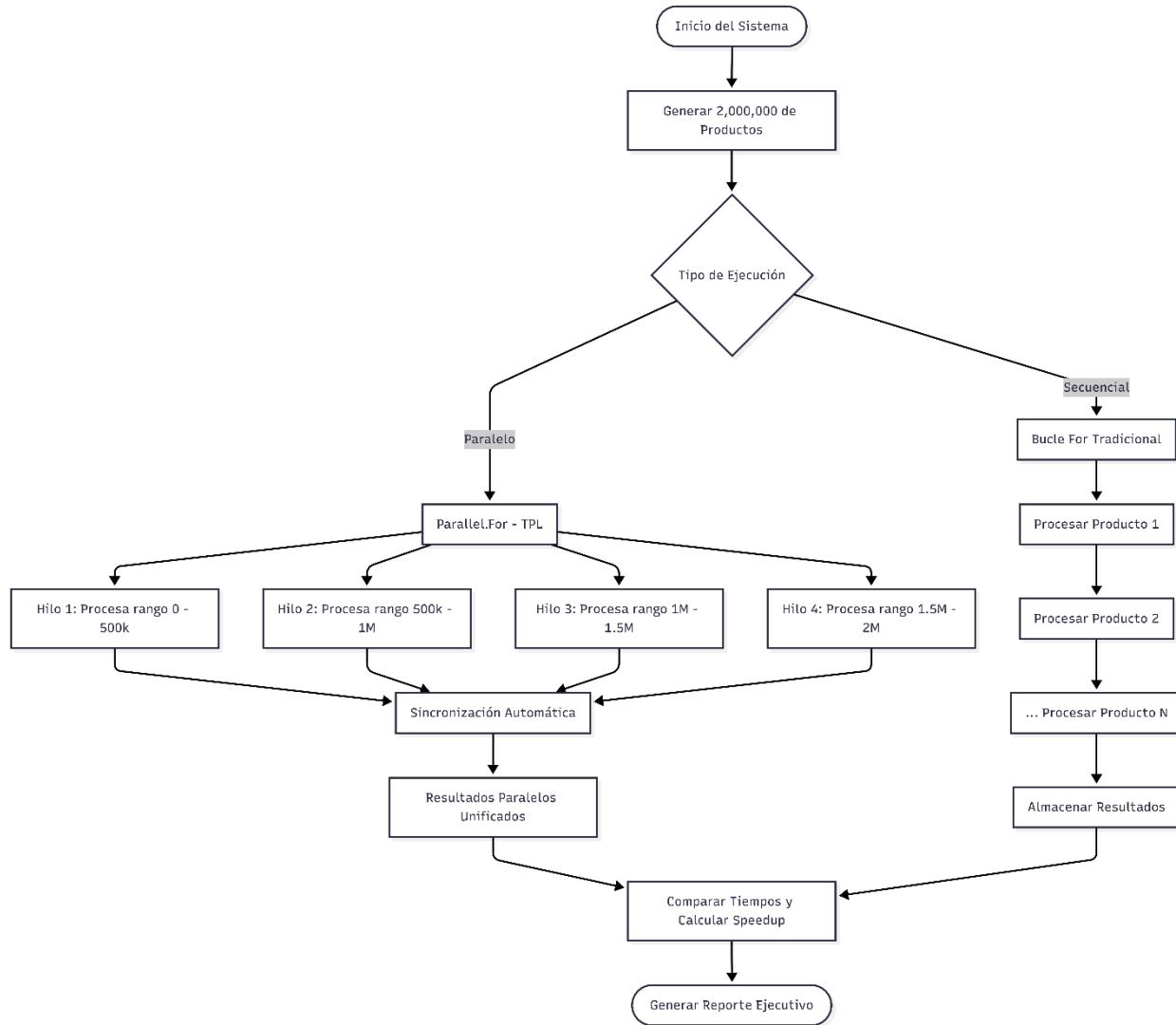
# 4. Diseño de la Solución

## Arquitectura general del sistema

Hemos diseñado el sistema siguiendo una arquitectura modular y limpia. La solución no es un solo bloque de código, sino que está dividida en cuatro componentes principales que interactúan entre sí:

1. **Generador de Datos (InventarioDatos):** Responsable de crear el dataset, No cargamos un excel externo para no limitar la velocidad por lectura de disco; generamos los datos en memoria ram para asegurar que el cuello de botella sea solo de procesador.
2. **Motor de Procesamiento (MotorEspeculativo):** Es el cerebro lógico. Recibe un producto y devuelve una predicción.
3. **Orquestador (Program):** Es el punto de entrada que coordina los tiempos, ejecuta las pruebas secuenciales y paralelas, y mide el rendimiento.
4. **Analista de Resultados (AnalistaReportes):** Toma los millones de resultados y los resume en información útil para la toma de decisiones.

## Diagrama de componentes/tareas paralelas



## Estrategia de paralelización utilizada

La estrategia central del proyecto es el Paralelismo Especulativo. A diferencia del procesamiento de datos tradicional donde se busca una única respuesta determinista nuestro sistema debe explorar múltiples futuros posibles al mismo tiempo. Para cada uno de los millones de productos, el algoritmo especula tres caminos distintos:

1. **Escenario Optimista:** Predicción de aumento de demanda.
2. **Escenario Pesimista:** Predicción de caída de ventas.
3. **Escenario Neutro:** Mantenimiento de la tendencia.

El paralelismo es fundamental aquí porque multiplicamos la carga de trabajo: no procesamos millones de datos simples, sino millones de escenarios hipotéticos 3 por producto. Utilizamos Parallel.For para distribuir esta carga masiva de especulación entre todos los núcleos disponibles. Esto nos permite simular qué pasaría en millones de productos en cuestión de milisegundos.

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

**Lenguaje:** C# .NET 8.0

**Librería TPL (Task Parallel Library):** Utilizamos System.Threading.Tasks para gestionar la creación de hilos y las tareas especulativas.

**Métricas de Tiempo:** System.Diagnostics.Stopwatch para medir con precisión de milisegundos el impacto de la ejecución especulativa paralela frente a la secuencial.

## 5. Implementación Técnica

### Descripción de la estructura del proyecto

El proyecto está construido bajo una estructura de consola en C#, organizado en clases con responsabilidades únicas para facilitar el trabajo en equipo del grupo. No pusimos todo el código en el Main, sino que lo sepáramos así:

- **Program.cs:** Contiene el flujo principal. Aquí definimos la constante CANTIDAD\_PRODUCTOS = 2000000 y orquestamos la comparación entre el modo secuencial y el paralelo.
- **InventarioDatos.cs:** Se encarga de la generación del dataset. Usa paralelismo incluso para crear los datos, asegurando que la preparación del entorno no sea lenta.
- **MotorEspeculativo.cs:** Contiene la lógica pura de negocio. Es donde ocurren los cálculos matemáticos de los escenarios de subida, bajada y estabilidad.
- **AnalistaReportes.cs:** Toma el arreglo final de resultados y usa LINQ para agrupar y contar tendencias, generando el reporte visual en consola.

### Explicación del código clave

El sistema se apoya en un conjunto de métodos centrales que permiten cumplir con los objetivos de generación masiva, simulación especulativa y análisis de tendencias. Dentro de estos métodos podemos encontrar abstracciones que coordinan el flujo de datos y la ejecución paralela:

- **InventarioDatos.Generar(int cantidad):** Se encarga de construir en memoria la base de datos simulada, utilizando internamente técnicas de paralelismo para crear millones de

registros de productos con atributos aleatorios como stock y ventas históricas, de manera eficiente, evitando que la preparación de datos sea un cuello de botella.

- **MotorEspeculativo.Procesar(Producto p):** Actúa como el núcleo lógico "stateless" (sin estado). Se encarga de recibir un producto individual y ejecutar la matemática especulativa para calcular tres escenarios futuros posibles (subida, bajada, estable), validarlos contra una simulación de demanda y devolver el escenario ganador sin interferir con otros hilos de ejecución.
- **AnalistaReportes.MostrarInformeEjecutivo(ResultadoPrediccion[] resultados):** Se encarga de recibir el conjunto masivo de predicciones procesadas y aplicar consultas LINQ para agrupar los datos por categoría. Su función es transformar millones de cálculos crudos en un reporte consolidado que muestra conteos y tendencias claras para la toma de decisiones.

### Uso de mecanismos de sincronización

Curiosamente, la mejor decisión que tomamos fue no usar locks en la fase crítica:

Porque los bloqueos detienen los hilos y matan el rendimiento.

Usamos un enfoque de escritura por índice. Antes de procesar, preasignamos un arreglo vacío del tamaño exacto: ResultadoPrediccion[] resPar = new ResultadoPrediccion[CANTIDAD\_PRODUCTOS];. Como cada hilo del Parallel.For recibe un índice **i** único, cada hilo escribe en su propia casilla del arreglo resPar[i]. Ningún hilo choca con otro porque nadie intenta escribir en la misma posición a la vez. Esto elimina la necesidad de lock, haciendo que el código sea extremadamente rápido.

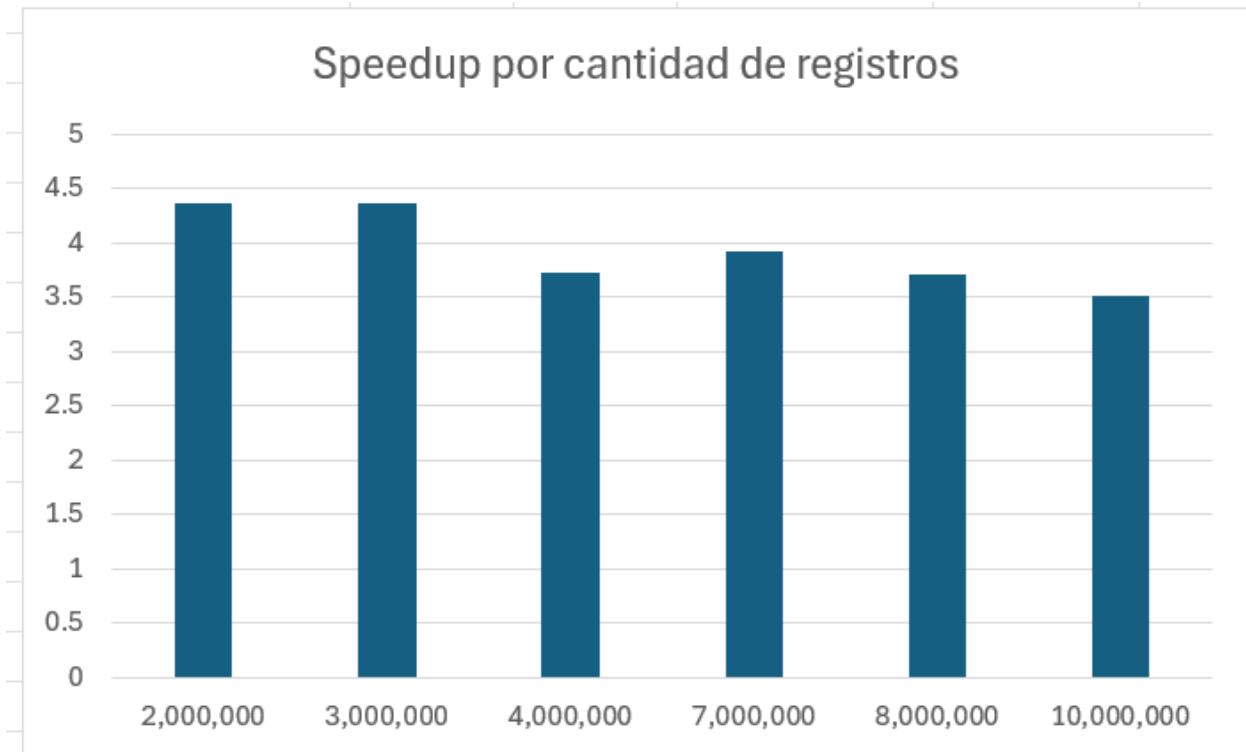
### Justificación técnica de las decisiones tomadas

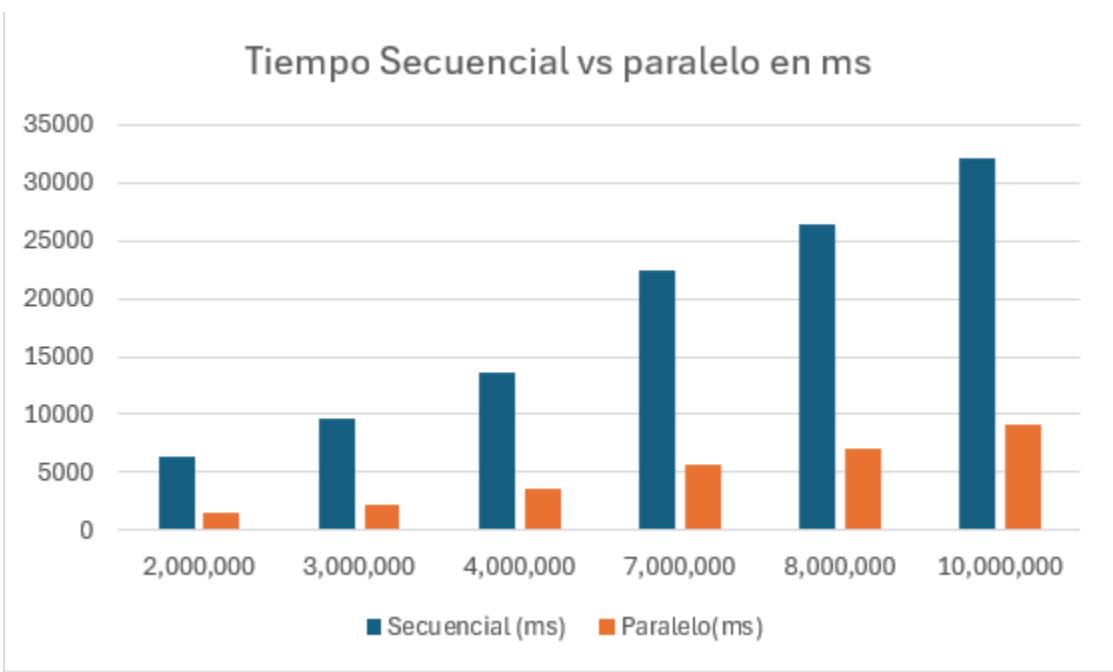
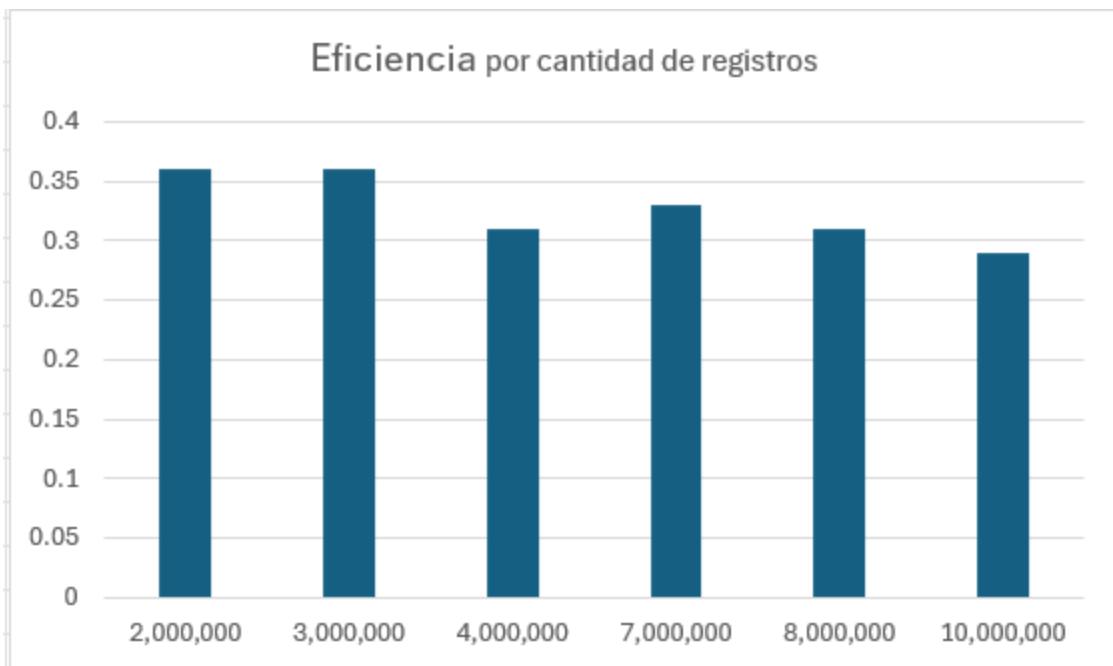
**Arrays vs Listas:** Usamos Arreglos en lugar de Listas para el almacenamiento de resultados. Las listas no son seguras para hilos al agregar elementos simultáneamente y requieren bloqueos. Los arreglos, al tener tamaño fijo, permiten acceso directo y seguro por índice.

**Stateless Methods:** Hicimos que el método Procesar fuera estático y no dependiera de variables globales. Esto permite que cualquier hilo lo llame sin riesgo de alterar el estado de otros hilos.

## 6. Evaluación de Desempeño

Cantidad de registros	Speedup	Eficiencia	Secuencial (ms)	Paralelo(ms)
2,000,000	4.36	0.36	6417	1473
3,000,000	4.37	0.36	9573	2192
4,000,000	3.73	0.31	13640	3657
7,000,000	3.92	0.33	22417	5724
8,000,000	3.71	0.31	26395	7114
10,000,000	3.51	0.29	32110	9149





## 7. Trabajo en Equipo

**Descripción del reparto de tareas**

Título	Asignado a	Descripción
<b>feat:</b> <b>Estructura y Orquestación</b>	<b>Wilmar Sanchez</b> (Arquitecto)	Crear el Program.cs, configurar el Stopwatch y las llamadas a los módulos. Responsable de medir el Speedup.
<b>feat: Lógica de Negocio y Simulación</b>	<b>Berlyng Yena</b> (Matemático)	Crear MotorEspeculativo.cs. Implementar el algoritmo de predicción.
<b>feat: Gestión de Datos</b>	<b>Carlos Reynoso</b> (Data Manager)	Crear Modelos.cs y InventarioDatos.cs. Generar la lista de 2 millones de productos usando paralelismo.
<b>feat:</b> <b>Reportes</b>	<b>Elys Batista</b> (Analista)	Crear AnalistaReportes.cs. Recibir el array de resultados, agrupar con LINQ e imprimir el reporte con colores en consola.

#### Herramientas utilizadas para coordinación (Git, AzureDevops, Trello, Excel, etc.)

Usamos Git para rastrear todos los cambios que hacíamos. Esto fue vital porque en el código paralelo, un cambio pequeño puede romper todo el rendimiento.

Subimos todo a GitHub, que funcionó como nuestro repositorio central en la nube. Así, cada uno podía trabajar en su parte y luego fusionar el código sin problemas.

La división de tarea fue muy fácil y la hicimos en una tabla de Excel

## 8. Conclusiones

### Principales aprendizajes técnicos

Wilmar Sanchez:

Aunque ya conocía la librería TPL y el paralelismo por las clases, este proyecto me permitió profundizar realmente en su uso práctico. También mejoré mucho mi capacidad para orquestar código modular, asegurando que las clases de mis compañeros se integraran sin errores en el flujo principal del Program.cs.

Berlyng Manuel Yena Garcia:

Aprendizajes técnicos importantes que aprendí es el uso adecuado de constantes para controlar el comportamiento del sistema, la aplicación de estadísticas simples como promedio para generar predicciones, la implementación de simulaciones probabilísticas con Random, el modelado de comportamientos distintos por categoría mediante un switch, el uso del error absoluto para validar qué escenario es más preciso, y la correcta de un modelo estructurado que separa claramente la lógica de predicción de otras capas del sistema.

Carlos Manuel Reynoso Ramón:

Aprendí a diseñar y estructurar modelos de datos claros y coherentes, asegurando que cada entidad representara correctamente la información del sistema. También fortalecí mi capacidad para organizar y gestionar los datos utilizados por las operaciones paralelas, validando información, manteniendo integridad entre modelos y facilitando que otras capas del proyecto pudieran acceder y manipular los datos de forma segura y eficiente.

Elys Camila Batista Encarnación:

Este proyecto me permitió comprender de manera práctica como la programación paralela transforma el rendimiento de sistemas que manejan grandes volúmenes de datos. Aprendí a utilizar Task Parallel Library y métodos, como Parallel.For, entendiendo como distribuir tareas entre múltiples núcleos para reducir tiempos de ejecución.

Además, reforcé conceptos sobre arquitectura modular en c#, separación de responsabilidades y diseño de métodos Stateless, lo que garantiza seguridad en entornos multihilo. También adquirí experiencia en el uso de métricas como Speedup Y Eficiencia, que son esenciales para evaluar el impacto real del paralelismo en la optimización del software.

## **Retos enfrentados y superados**

Wilmar Sanchez:

Mi mayor desafío fue la coordinación técnica. Al ser quien unía el código, tuve que resolver varios conflictos al integrar la lógica de mis compañeros con el sistema de medición de tiempos. También me costó gestionar como funcionaría todo integrado para que se notara la mejora del paralelismo en relación a la cantidad de datos.

Berlyng Manuel Yena Garcia:

Uno de los principales retos que enfrenté fue lograr que la simulación se comportara de manera coherente para cada categoría, ya que al principio los factores de subida y bajada generaban resultados poco realistas; para superarlo tuve que ajustar las constantes y entender mejor cómo influían estadísticamente en el promedio. También me costó manejar correctamente la aleatoriedad, porque quería que cada producto tuviera una variación propia, y en ese proceso aprendí la importancia de controlar la semilla y el uso adecuado de Random. Otro reto fue

comparar los escenarios especulativos contra la demanda simulada de forma justa, hasta que finalmente comprendí que el error absoluto era la mejor forma de medir la distancia entre predicciones y realidad. Al final, estos desafíos me ayudaron a entender mejor cómo estructurar lógica compleja, cómo organizar código más limpio y cómo validar decisiones basadas en datos.

Carlos Manuel Reynoso Ramón:

Unos de los principales desafíos fue pasar de una generación básica de la lista de productos a una versión más eficiente utilizando procesamiento paralelo. Al realizar esta mejora, surgieron retos como garantizar que los valores aleatorios fueran consistentes entre hilos

Elys Camila Batista Encarnación:

Uno de los principales desafíos fue evitar condiciones de carrera al trabajar con todos compartidos. Inicialmente considere usar bloqueos (lock), pero descubri que esto afectaba negativamente el rendimiento. La solucion fue implementar escritura por indice en arreglos preasignados, lo que me enseño la importancia de diseñar algoritmos seguros para hilos sin sacrificar velocidad. Otro reto fue comprender como generar datos masivos sin que la preparacion se convirtiera en un cuello de botella; para ello, aplicamos paralelismo incluso en la fase de generacion del inventario. Finalmente, interpretar y validar la coherencia de las predicciones me llevo a profundizar en análisis estadístico y control de aleatoriedad.

## **Possibles mejoras o líneas futuras**

Wilmar Sanchez:

Para el futuro, me gustaría hacer el sistema interactivo para que no tenga valores fijos en código. También sería bueno implementar una barra de carga visual para mejorar la experiencia de usuario y exportar los reportes finales a un archivo Excel real en lugar de solo mostrarlos en consola.

Berlyng Manuel Yena Garcia:

Al proyecto se le pueden hacer varias mejoras futuras, como implementar un generador de números aleatorios con semilla controlada para que las simulaciones sean reproducibles, ajustar los factores por categoría usando datos reales en lugar de valores fijos, y añadir más escenarios especulativos basados en tendencias históricas. También sería útil convertir la lógica en un servicio más modular para permitir pruebas unitarias más precisas, optimizar el manejo de categorías usando enums o un diccionario de reglas, y expandir el modelo de salida para incluir métricas como el porcentaje de error o la confianza de la predicción. Estas mejoras harían el sistema más realista, escalable y fácil de mantener.

Carlos Manuel Reynoso Ramón:

Entre las mejoras que podríamos considerar está mejorar la calidad de los datos generados, optimizar cómo se manejan los valores aleatorios, permitir obtener productos desde fuentes externas y agregar pruebas automatizadas que garanticen la estabilidad del proceso en diferentes condiciones.

Elys Camila Batista Encarnación:

El sistema puede evolucionar significativamente. Una mejora clave sería incorporar datos reales para ajustar los factores de predicción por categoría, logrando resultados más precisos. También sería útil implementar un generador aleatorio con semilla controlada para garantizar reproducibilidad en las simulaciones. Otra línea futura es convertir la aplicación en un servicio web escalables, integrable con plataformas empresariales, y añadir pruebas unitarias para asegurar calidad y mantenibilidad. Además, se podrían incluir más escenarios especulativos basados en tendencias históricas y métricas avanzadas como porcentaje de error y nivel de confianza, lo que haría el sistema más robusto y aplicables en entornos reales.

## 9. Referencias

### Fuentes bibliográficas, técnicas o académicas consultadas

**Microsoft Learn.** (2024). *Parallel Programming in .NET: A guide to the documentation*. Recuperado de: <https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/>

**Microsoft Learn.** (2024). *Task Parallel Library (TPL)*. Recuperado de: <https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl>

**Microsoft Learn.** (2024). *Parallel Class (System.Threading.Tasks)*. Recuperado de: <https://learn.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel>

## 10. Anexos

### 1. Requisitos Indispensables

Asegúrese de tener instalado el **.NET SDK** (la herramienta principal para compilar y ejecutar el proyecto) y **Git**.

### 2. Preparación y Descarga del Código

Abra el **CMD** o su terminal preferida, y siga estos pasos:

#### 1. Clonar el Repositorio:

```
Símbolo del sistema Microsoft Windows [Versión 10.0.26100.7171]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Reynoso Taveras>git clone https://github.com/wilmar0312/SistemaDeInventarioEspeculativo
Cloning into 'SistemaDeInventarioEspeculativo'...
remote: Enumerating objects: 100, done.
remote: Counting objects: 100% (100/100), done.
remote: Compressing objects: 100% (74/74), done.
remote: Total 100 (delta 44), reused 65 (delta 23), pack-reused 0 (from 0)
Receiving objects: 100% (100/100), 22.80 KiB | 686.00 KiB/s, done.
Resolving deltas: 100% (44/44), done.

C:\Users\Reynoso Taveras>
```

## 2. Acceder al Directorio del Proyecto:

```
Símbolo del sistema Microsoft Windows [Versión 10.0.26100.7171]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Reynoso Taveras>cd SistemaDeInventarioEspeculativo
C:\Users\Reynoso Taveras\SistemaDeInventarioEspeculativo>cd src
C:\Users\Reynoso Taveras\SistemaDeInventarioEspeculativo\src>cd proyectoParaleloFinal
C:\Users\Reynoso Taveras\SistemaDeInventarioEspeculativo\src\ProyectoParaleloFinal>
```

## 3 . Ejecutar el programa con dotnet run

```

C:\Users\Reynoso Taveras\SistemaDeInventarioEspeculativo\src\ProyectoParaleloFinal>dotnet run
C:\Users\Reynoso Taveras\SistemaDeInventarioEspeculativo\src\ProyectoParaleloFinal\Modelos.cs(6,23): warning CS8618: El elemento propiedad "Categoria" que no acepta valores NULL debe contener un valor distinto de NULL al salir del constructor. Considere la posibilidad de agregar el modificador "required" o declarar el propiedad como un valor que acepta valores NULL.
C:\Users\Reynoso Taveras\SistemaDeInventarioEspeculativo\src\ProyectoParaleloFinal\Modelos.cs(14,23): warning CS8618: El elemento propiedad "Categoria" que no acepta valores NULL debe contener un valor distinto de NULL al salir del constructor. Considere la posibilidad de agregar el modificador "required" o declarar el propiedad como un valor que acepta valores NULL.
C:\Users\Reynoso Taveras\SistemaDeInventarioEspeculativo\src\ProyectoParaleloFinal\Modelos.cs(8,25): warning CS8618: El elemento propiedad "VentasUltimosDias" que no acepta valores NULL debe contener un valor distinto de NULL al salir del constructor. Considere la posibilidad de agregar el modificador "required" o declarar el propiedad como un valor que acepta valores NULL.
C:\Users\Reynoso Taveras\SistemaDeInventarioEspeculativo\src\ProyectoParaleloFinal\Modelos.cs(15,23): warning CS8618: El elemento propiedad "EscenarioGanador" que no acepta valores NULL debe contener un valor distinto de NULL al salir del constructor. Considere la posibilidad de agregar el modificador "required" o declarar el propiedad como un valor que acepta valores NULL.
C:\Users\Reynoso Taveras\SistemaDeInventarioEspeculativo\src\ProyectoParaleloFinal\MotorEspeculativo.cs(122,13): warning CS0162: Se detectó código inaccesible
== SISTEMA INTEGRADO DE PREDICCIÓN PARALELA ==
Generando 2,000,000 registros...
-> Ejecutando Análisis SECUENCIAL...
Tiempo: 2372 ms
-> Ejecutando Análisis PARALELO...
Tiempo: 1946 ms
--- REPORTE EJECUTIVO POR CATEGORÍA ---
[Hogar] -> Tendencia: ESTABLE
Detalle: 0 suben, 0 bajan, 499995 estables.
[Limpieza] -> Tendencia: ALTA DEMANDA (REABASTECER)
Detalle: 250245 suben, 249688 bajan, 0 estables.
[Electrónica] -> Tendencia: BAJA DEMANDA (OFERTAR)
Detalle: 149886 suben, 350164 bajan, 0 estables.
[Alimentos] -> Tendencia: ALTA DEMANDA (REABASTECER)
Detalle: 349711 suben, 0 bajan, 150311 estables.
--- Metrcas de rendimiento ---
Speedup: 1.53x
Eficiencia: 0.38
Presione ENTER para finalizar...

```

#### 4. Resultado La consola ejecutará la aplicación.

- Aplicación de Consola: La consola mostrará directamente un reporte sobre los productos y las demandas de estos mismos.

#### Enlace al repositorio de Github (público)

<https://github.com/wilmar0312/SistemaDeInventarioEspeculativo.git>