

**Miembros:**

Manuel De Jesus Rivas Tavarez 20231660

Netanel De Jesús 20231103

Yerelin Vanessa Rosario Taveras 20231751

Jordi Jose Mena Rosario 20231186

Ricardo Alexander Diaz Santana 20240244

**Carrera:**

Desarrollo de Software

**Materia:**

Programación Paralela

**Nombre del docente:**

Erick Pérez

**Tema:**

Proyecto Final: Brigada de Emergencia

**Líder:**

Manuel De Jesus Rivas Tavarez

**Fecha:**

20/08/2025

**Índice del Proyecto "Brigadas de Emergencia Durante Tormenta Tropical"**

1. Introducción

* 1.1 Presentación general del proyecto
* 1.2 Justificación del tema elegido
* 1.3 Objetivos
  + 1.3.1 Objetivo general
  + 1.3.2 Objetivos específicos

2. Descripción del Problema

* 2.1 Contexto del problema seleccionado
* 2.2 Aplicación del problema en un escenario real
* 2.3 Importancia del paralelismo en la solución

3. Cumplimiento de los Requisitos del Proyecto

* 3.1 Ejecución simultánea de múltiples tareas
* 3.2 Necesidad de compartir datos entre tareas
* 3.3 Exploración de diferentes estrategias de paralelización
* 3.4 Escalabilidad con más recursos
* 3.5 Métricas de evaluación del rendimiento
* 3.6 Aplicación a un problema del mundo real

4. Diseño de la Solución

* 4.1 Arquitectura general del sistema
* 4.2 Diagrama de componentes/tareas paralelas
* 4.3 Estrategia de paralelización utilizada
* 4.4 Herramientas y tecnologías empleadas (C#, TPL, etc.)

5. Implementación Técnica

* 5.1 Descripción de la estructura del proyecto
* 5.2 Explicación del código clave
* 5.3 Uso de mecanismos de sincronización
* 5.4 Justificación técnica de las decisiones tomadas

6. Evaluación de Desempeño

* 6.1 Comparativa entre ejecución secuencial y paralela
* 6.2 Métricas: tiempo de ejecución, eficiencia, escalabilidad
* 6.3 Gráficas o tablas con resultados
* 6.4 Análisis de cuellos de botella o limitaciones
* 6.5 Mejoras sugeridas

7. Trabajo en Equipo

* 7.1 Descripción del reparto de tareas
* 7.2 Herramientas utilizadas para coordinación (Git, AzureDevops, Trello, Excel, etc.)

8. Conclusiones

* 8.1 Principales aprendizajes técnicos
* 8.2 Retos enfrentados y superados
* 8.3 Posibles mejoras o líneas futuras

9. Referencias

* 9.1 Fuentes bibliográficas, técnicas o académicas consultadas

10. Anexos

* 10.1 Manual de ejecución del sistema
* 10.2 Enlace al repositorio de Git (público)

**1. Introducción**

**Presentación general del proyecto**

El proyecto "Brigadas de Emergencia Durante Tormenta Tropical" consiste en el desarrollo de un sistema de simulación computacional en C# para modelar la respuesta de brigadas de emergencia en escenarios de tormentas tropicales en la República Dominicana. El sistema utiliza datos geográficos reales de provincias, municipios y barrios (basados en la Oficina Nacional de Estadística - ONE), incorporando paralelismo para optimizar el procesamiento de eventos aleatorios de emergencias como inundaciones y deslizamientos. La simulación incluye asignación de brigadas, métricas de rendimiento y configuración ajustable, con el objetivo de proporcionar una herramienta para planificación y entrenamiento en situaciones de desastre. El enfoque integra bibliotecas como Newtonsoft.Json para manejo de datos y Task Parallel Library (TPL) para ejecución concurrente, permitiendo comparaciones entre modos secuencial y paralelo.

**Justificación del tema elegido**

La República Dominicana es uno de los países más vulnerables a tormentas tropicales en el Caribe, con un historial de impactos devastadores que afectan la economía, infraestructura y vidas humanas. Según datos históricos, el país ha sido golpeado por al menos 45 huracanes desde 1851, con picos en meses como septiembre y octubre. Eventos recientes, como el huracán Fiona en 2022 que causó inundaciones masivas en el este del país, y Beryl en 2024 que afectó el sur, destacan la necesidad de sistemas predictivos para respuesta rápida. Barrios urbanos densos, como los de Santo Domingo Este (donde el 45% de la población enfrenta riesgos anuales de inundaciones), son particularmente expuestos, con vulnerabilidad agravada por factores socioeconómicos y geográficos. Este tema se justifica por la frecuencia de desastres (e.g., Irma y Maria en 2017 causaron pérdidas millonarias y desplazamientos), la utilidad de simulaciones para mitigar riesgos, y la oportunidad de aplicar computación paralela para procesar datos a gran escala, mejorando la eficiencia en escenarios reales de emergencia.

**Objetivos (general y específicos)**

**Objetivo general:** Desarrollar un sistema de simulación paralelo en C# para modelar la respuesta de brigadas de emergencia durante tormentas tropicales en la República Dominicana, utilizando datos geográficos reales para optimizar la asignación de recursos y medir el rendimiento.

**Objetivos específicos:**

* Cargar y estructurar datos de provincias, municipios y barrios de RD desde archivos JSON para representar escenarios vulnerables a inundaciones.
* Implementar un simulador de eventos aleatorios de emergencias, integrando paralelismo para procesar múltiples incidentes simultáneamente y reducir tiempos de ejecución.
* Evaluar métricas de rendimiento, comparando modos secuencial y paralelo, para demostrar la importancia de la computación concurrente en simulaciones de desastres.
* Aplicar el sistema a un escenario real basado en tormentas históricas, como Fiona (2022), para validar su utilidad en planificación de respuestas.

**2. Descripción del Problema**

**Contexto del problema seleccionado**

La República Dominicana enfrenta un alto riesgo de tormentas tropicales y huracanes debido a su ubicación en el Caribe, con la temporada anual de junio a noviembre trayendo lluvias intensas, vientos fuertes y inundaciones. Datos históricos muestran impactos frecuentes: desde 1851, el país ha experimentado huracanes como David (1979, categoría 5) y George (1998), hasta eventos recientes como Matthew (2016), Irma y Maria (2017), y Fiona (2022), que causaron evacuaciones masivas y daños en infraestructuras. Barrios en áreas bajas, como los de Santo Domingo (donde el 37% de la población vive en zonas costeras vulnerables a tsunamis e inundaciones), son particularmente afectados, con demografías que incluyen comunidades de bajos ingresos expuestas a riesgos anuales del 45% en municipios como Santo Domingo Este. La vulnerabilidad se agrava por factores socioeconómicos (e.g., barrios con minorías étnicas y bajos recursos tienen mayor exposición a floods) y geográficos (e.g., ríos Ozama e Isabela en la capital incrementan riesgos), haciendo esencial simular respuestas para mitigar pérdidas humanas y económicas.

**Aplicación del problema en un escenario real**

En un escenario real, como el huracán Fiona (2022), que golpeó el este de RD con vientos de 150 km/h y lluvias que causaron inundaciones en barrios de Samaná, Hato Mayor y Santo Domingo, el sistema simularía la asignación de brigadas a zonas afectadas. Por ejemplo, barrios como Sabana Perdida (Santo Domingo Norte) o Los Mameyes (Santo Domingo Este), con alta densidad poblacional y vulnerabilidad a floods (demografías muestran ~175K habitantes expuestos en estas áreas), requerirían respuesta rápida para evacuaciones y rescates. La simulación modelaría eventos aleatorios (e.g., inundaciones en ríos Ozama), usando datos geográficos para priorizar barrios costeros o de bajos recursos, donde arrecifes naturales reducen riesgos, pero no eliminan la necesidad de brigadas eficientes. En práctica, ayudaría a autoridades como COE (Centro de Operaciones de Emergencias) a planificar, similar a cómo se usó en Beryl (2024) para alertas tempranas.

**Importancia del paralelismo en la solución**

El paralelismo es crucial en simulaciones de emergencias para procesar grandes volúmenes de datos en tiempo real, acelerando cálculos como asignación de brigadas en múltiples barrios simultáneamente. En escenarios de tormentas, donde eventos ocurren concurrentemente (e.g., inundaciones en varios municipios), el procesamiento secuencial es ineficiente, pero técnicas como TPL permiten simular miles de incidentes hasta 5 veces más rápido. Por ejemplo, en modelos de dispersión atmosférica o evacuación por tsunamis, el paralelismo reduce tiempos de ejecución, facilitando diseños optimizados para respuestas rápidas en RD, donde barrios vulnerables requieren predicciones instantáneas para minimizar pérdidas. Esto mejora la resiliencia, permitiendo experimentos computacionales en sociedades artificiales para escenarios no convencionales.

## **3. Cumplimiento de los Requisitos del Proyecto**

### **1. Ejecución simultánea de múltiples tareas**

El sistema implementa concurrencia real mediante la ejecución paralela de múltiples procesos. Durante la simulación en vivo, el sistema ejecuta simultáneamente tres tareas principales: la generación continua de emergencias, el procesamiento individual de cada emergencia por brigadas disponibles, y la actualización constante de la interfaz de usuario con estadísticas en tiempo real. Cada emergencia que se genera se procesa de manera asíncrona e independiente, permitiendo que múltiples brigadas atiendan diferentes emergencias al mismo tiempo. El gestor paralelo coordina estas tareas utilizando el pool de hilos del sistema operativo para maximizar la utilización de recursos.

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

El proyecto requiere compartir información crítica entre múltiples hilos de ejecución. Las brigadas mantienen su estado de disponibilidad que debe ser consultado y modificado por diferentes tareas simultáneamente. Las estadísticas del sistema como el número de emergencias generadas, atendidas y brigadas en servicio se actualizan desde múltiples hilos concurrentes. La lista de emergencias en cola es accedida por el generador de emergencias y los procesadores de manera concurrente. Para garantizar la integridad de estos datos compartidos, el sistema implementa mecanismos de sincronización mediante operaciones atómicas para contadores y bloqueos para operaciones más complejas como la escritura a consola.

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

El proyecto explora múltiples enfoques de paralelización. Implementa paralelismo de datos mediante el procesamiento simultáneo de listas de emergencias, donde cada elemento se procesa independientemente en hilos separados. Utiliza paralelismo de tareas para ejecutar operaciones conceptualmente diferentes de manera concurrente, como la generación de emergencias y el monitoreo del sistema. El sistema permite configurar dinámicamente el grado de paralelismo, adaptándose desde ejecución secuencial hasta utilizar todos los núcleos disponibles del procesador. Además, implementa un sistema de gestión de recursos limitados donde las brigadas actúan como recursos compartidos que deben ser asignados eficientemente.

### **4. Escalabilidad con más recursos**

El sistema está diseñado para escalar con el hardware disponible. Incluye un módulo completo de análisis de speedup que prueba sistemáticamente diferentes configuraciones de núcleos, desde ejecución con un solo hilo hasta utilizar la capacidad máxima del procesador. Mide el rendimiento con cada configuración y calcula métricas de eficiencia para determinar el punto óptimo de utilización de recursos. El sistema detecta automáticamente cuando agregar más núcleos ya no proporciona beneficios proporcionales, identificando el punto de rendimientos decrecientes. Esta funcionalidad permite a los usuarios determinar la configuración óptima para su hardware específico.

### **5. Métricas de evaluación del rendimiento**

El proyecto implementa un sistema completo de medición y análisis de rendimiento. Mide con precisión los tiempos de ejecución tanto en modo secuencial como paralelo utilizando cronómetros de alta resolución. Calcula el speedup como la razón entre el tiempo secuencial y paralelo, proporcionando una medida directa de la mejora obtenida. Determina la eficiencia como el porcentaje de utilización efectiva de los recursos, ayudando a identificar si el paralelismo está siendo aprovechado óptimamente. Mide el throughput en términos de emergencias procesadas por segundo. Adicionalmente, recopila estadísticas detalladas sobre el comportamiento del paralelismo, incluyendo el número de hilos concurrentes activos, el balance de carga entre hilos, y la distribución de tareas.

### **6. Aplicación a un problema del mundo real**

El proyecto aborda un problema genuino de gestión de emergencias en República Dominicana. Utiliza datos geográficos reales del país, incluyendo todas las provincias, municipios y barrios, con sus coordenadas geográficas precisas. Modela tipos de emergencias específicos y relevantes para la región, como inundaciones, deslizamientos de tierra, y eventos relacionados con huracanes, considerando la vulnerabilidad climática particular de cada provincia. El sistema simula brigadas de rescate con capacidades y ubicaciones realistas, implementando algoritmos de asignación basados en proximidad geográfica y disponibilidad de recursos. La optimización del tiempo de respuesta mediante paralelización tiene implicaciones directas en la vida real, donde reducir el tiempo de coordinación de emergencias puede significar la diferencia entre salvar vidas o no. El análisis de rendimiento ayuda a determinar la configuración óptima de centros de comando y sistemas de despacho para maximizar la eficiencia operacional con los recursos disponibles.

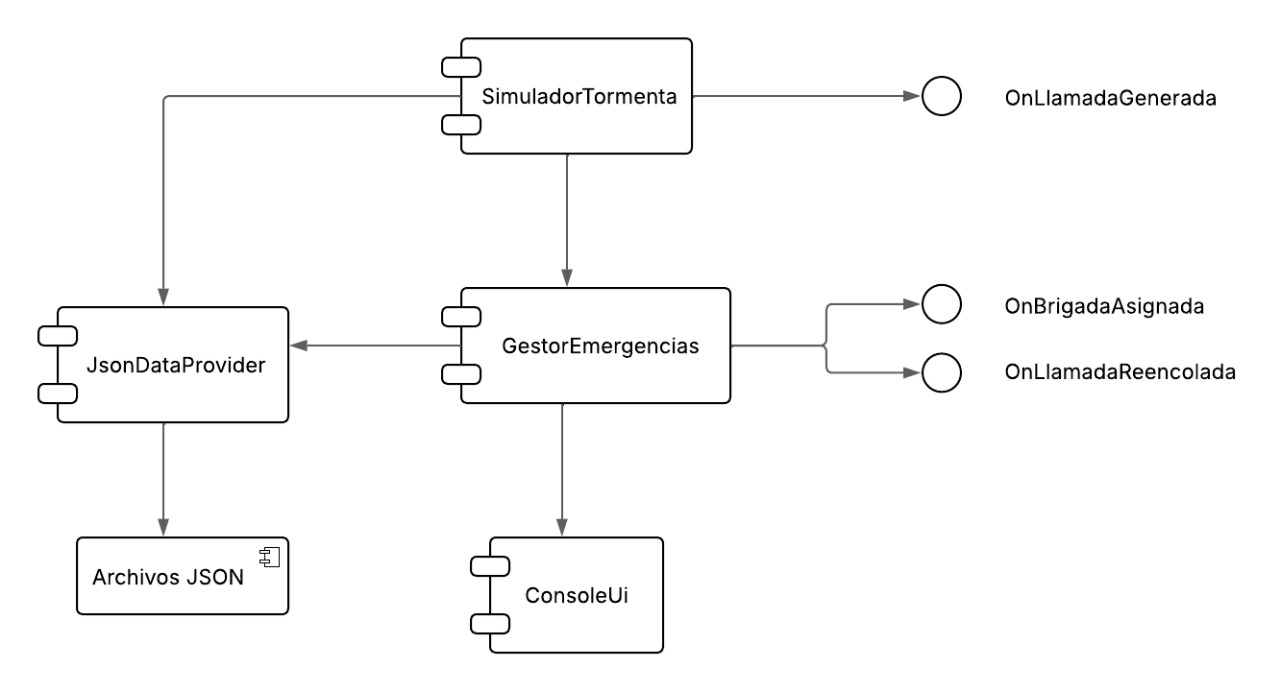
**4. Diseño de la Solución**

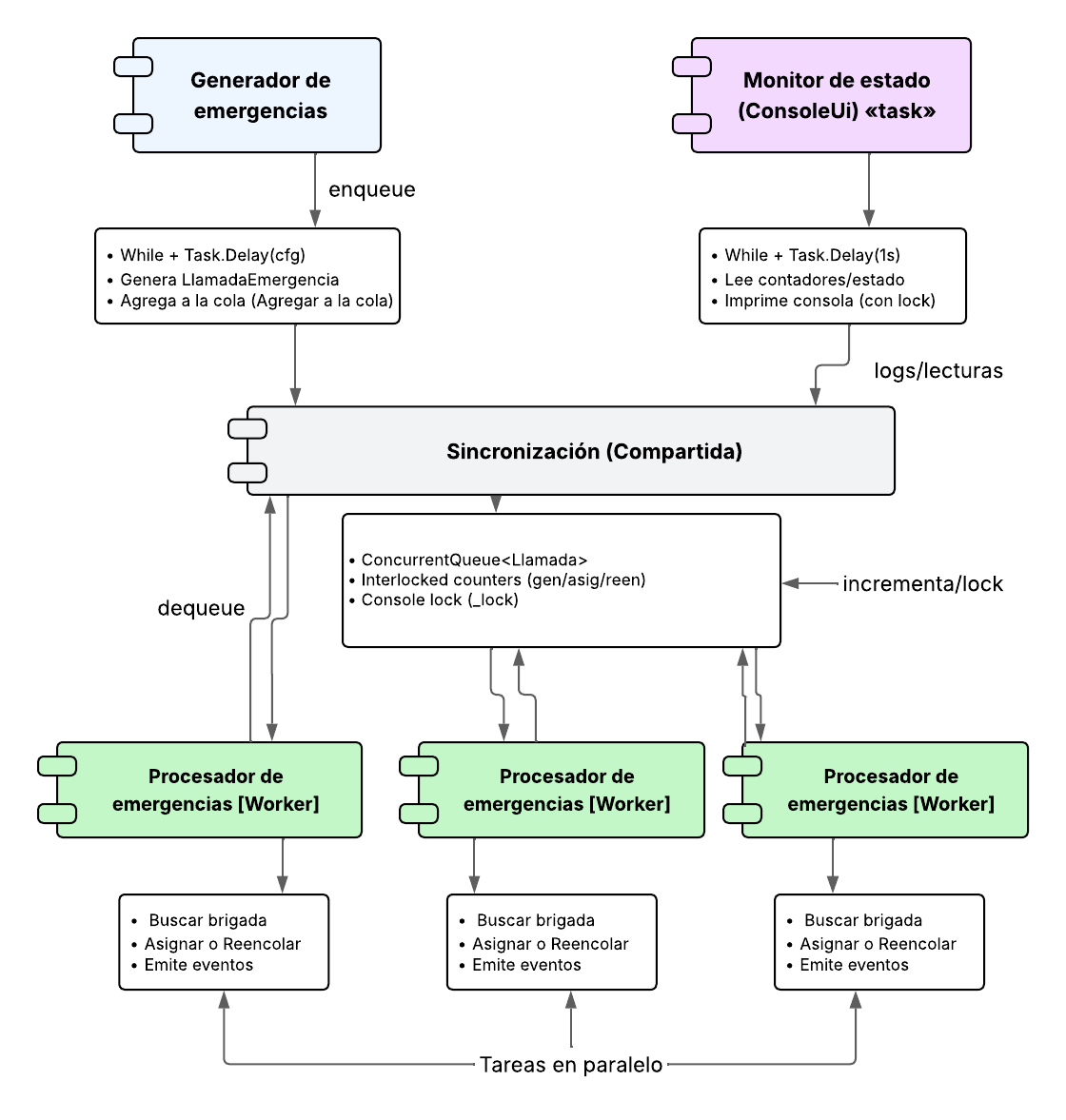
1. **Arquitectura general del sistema**

El sistema sigue una arquitectura modular de tres capas principales. La capa de datos maneja la información geográfica de República Dominicana y las brigadas disponibles mediante el patrón Repository implementado en JsonDataProvider. La capa de lógica de negocio contiene los modelos de dominio (Provincia, Municipio, Barrio, Brigada, EmergenciaEvento) y las reglas de asignación de recursos. La capa de presentación incluye la interfaz de consola con simulaciones en tiempo real y análisis de rendimiento.

La arquitectura implementa el patrón Producer-Consumer para la generación y procesamiento de emergencias. Un generador produce emergencias de manera continua mientras múltiples consumidores (brigadas) las procesan concurrentemente. El sistema utiliza un coordinador central que gestiona el estado compartido y sincroniza las operaciones entre componentes.

1. **Diagrama de componentes/tareas paralelas**

****

**Tareas paralelas:   
**

1. **Estrategia de paralelización utilizada**

El proyecto emplea paralelización híbrida combinando múltiples estrategias. Para operaciones de datos utiliza paralelismo de datos mediante Parallel.ForEach, distribuyendo listas de emergencias entre hilos disponibles. Para operaciones concurrentes implementa paralelismo de tareas usando async/await y Task.Run.

La estrategia incluye control granular del grado de paralelismo mediante ParallelOptions.MaxDegreeOfParallelism, permitiendo ajustar la utilización de recursos según el hardware disponible. Implementa un sistema de cancelación cooperativa usando CancellationToken para terminar operaciones limpiamente.

Para recursos limitados como brigadas, utiliza un patrón de semáforo implícito donde el estado de disponibilidad actúa como control de acceso. La sincronización se maneja mediante operaciones atómicas para contadores simples y bloqueos explícitos para operaciones complejas.

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

**C# y .NET**: Plataforma base que proporciona el runtime y las librerías fundamentales del sistema.

**Task Parallel Library (TPL)**: Framework principal para paralelización, proporcionando Task, Parallel.ForEach, async/await y mecanismos de cancelación.

**System.Threading**: Utilizado para operaciones atómicas (Interlocked), tokens de cancelación (CancellationToken), y primitivas de sincronización.

**System.Diagnostics.Stopwatch**: Para medición precisa de tiempos de ejecución y cálculos de rendimiento.

**Newtonsoft.Json** (implícito en JsonDataProvider): Para deserialización de datos geográficos desde archivos JSON.

**5. Implementación Técnica**

1. **Descripción de la estructura del proyecto**

El proyecto se organiza en una estructura de capas bien definida. La carpeta Core contiene los modelos de dominio, enumeraciones, e interfaces que definen los contratos del sistema. La carpeta Data implementa el acceso a datos mediante el patrón Repository. La carpeta Parallelism encapsula toda la lógica de procesamiento concurrente.

La clase Program actúa como punto de entrada y controlador principal, coordinando las diferentes funcionalidades. Los métodos están organizados por regiones funcionales: simulación en vivo, simulaciones comparativas, análisis de rendimiento, y utilidades generales.

La separación de responsabilidades es clara: los modelos encapsulan datos y comportamiento de dominio, los providers manejan persistencia, los gestores paralelos coordinan concurrencia, y la interfaz de usuario maneja presentación e interacción.

1. **Explicación del código clave**

El núcleo del sistema paralelo reside en la función GenerarEmergenciasEnVivoAsync que implementa el patrón productor. Utiliza un bucle continuo con Task.Delay para generar emergencias a intervalos regulares. Cada emergencia se procesa mediante Task.Run, creando una nueva tarea que opera independientemente.

La función ProcesarEmergenciaEnVivoAsync implementa el patrón consumidor. Busca brigadas disponibles usando LINQ, actualiza estados mediante operaciones thread-safe, y programa la liberación de recursos usando una tarea diferida. Este diseño evita bloqueos prolongados y maximiza la concurrencia.

El gestor GestorParaleloExtendido encapsula la lógica de paralelización usando Parallel.ForEach con configuración personalizable. Implementa métricas detalladas recopilando información sobre hilos utilizados, tareas ejecutadas y balance de carga.

La función de análisis de speedup implementa un algoritmo sistemático que prueba diferentes configuraciones de núcleos, mide tiempos de ejecución, y calcula métricas de eficiencia. Esto proporciona datos cuantitativos sobre el comportamiento del paralelismo.

1. **Uso de mecanismos de sincronización**

El sistema implementa sincronización en múltiples niveles. Para contadores simples utiliza Interlocked.Increment y Interlocked.Decrement, proporcionando operaciones atómicas sin bloqueos que son eficientes para estadísticas concurrentes.

Para operaciones de consola utiliza lock con un objeto dedicado (\_lockConsole), asegurando que la escritura a pantalla sea coherente cuando múltiples hilos intentan mostrar información simultáneamente.

El estado de brigadas se protege mediante cambios atómicos de estado y programación de tareas de liberación. En lugar de usar locks explícitos, aprovecha la inmutabilidad de operaciones de lectura y la atomicidad de cambios de enumeración.

Para cancelación cooperativa utiliza CancellationToken propagado a través de todas las operaciones asíncronas, permitiendo terminación limpia de la simulación sin corrupción de estado.

1. **Justificación técnica de las decisiones tomadas**

La decisión de usar Task.Run para cada emergencia individual permite máxima concurrencia sin limitar el número de emergencias procesables simultáneamente. Esta aproximación es superior a usar un pool fijo de workers porque la carga es variable e impredecible.

El uso de Interlocked para estadísticas evita el overhead de locks mientras mantiene consistencia. Dado que son operaciones simples de incremento, la sincronización atómica es suficiente y más eficiente.

La implementación de múltiples estrategias de paralelización (datos vs tareas) permite evaluar diferentes casos de uso. Parallel.ForEach es óptimo para procesamiento batch mientras que async/await es mejor para operaciones I/O y coordinación.

La decisión de medir rendimiento con múltiples configuraciones de núcleos proporciona datos empíricos sobre escalabilidad real vs teórica. Esto es crucial porque el speedup teórico rara vez se alcanza debido a overhead de coordinación y limitaciones de recursos compartidos.

El diseño de liberación automática de brigadas mediante tareas programadas evita la necesidad de un garbage collector manual, simplificando la gestión de recursos mientras mantiene realismo en la simulación.

**6. Evaluación de Desempeño**

**1. Comparativa entre ejecución secuencial y paralela**

**Propósito.**

Evaluar el rendimiento del sistema de brigadas en dos modos de ejecucion: secuencial (procesa una emergencia a la vez) y paralelo (TPL), que distribuye emergencias entre varios hilos de trabajo. El objetivo es cuantificar tiempo de ejecucion, throughput (atenciones/seg), speedup, eficiencia y escalabilidad (comportamiento al aumentar los hilos), e identificar cuellos de botella.

**Puntos clave / archivos**

* Menu con opciones 2–5 en src/Program.cs.
* Reportes en texto plano generados en resultados-reportes/.
* Datos reales desde src/data/.
* **Datos**: se usan los JSON reales del proyecto (provincias, municipios, barrios, brigadas) para generar emergencias representativas.
* **Ejecuciones medidas** (todas con la **misma cantidad** de emergencias, p.ej. 200):  
  + **Opción 2**: Simulación secuencial con datos reales.
  + **Opción 3**: Simulación paralela con datos reales (Task Parallel Library).
  + **Opción 4**: Comparación secuencial vs paralela (calcula speedup/eficiencia).
  + **Opcion 5**: Análisis de speedup con múltiples núcleos (1, 2, 4, 8 y CPU).
* Paralelismo: MaxGradoParalelismo = Environment.ProcessorCount (hilos segun el equipo).
* Reportes: cada opción genera su .txt con tabla; nombres sin fecha y sobrescritura del ultimo resultado.

**Puntos clave / archivos**

* src/Program.cs (flujo y mediciones).
* resultados-reportes/sim\_secuencial\_reporte.txt
* resultados-reportes/sim\_paralela\_reporte.txt
* resultados-reportes/comparacion\_seq\_vs\_par\_reporte.txt
* resultados-reportes/speedup\_multinucleo\_reporte.txt

**2. Métricas: tiempo de ejecución, eficiencia, escalabilidad**

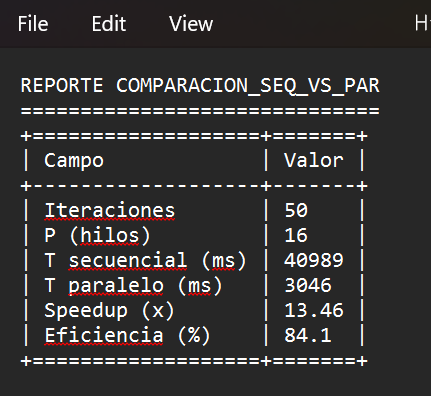
* Tiempo total (T): duración de la corrida.
* Throughput: atenciones por segundo = atendidas / Tsegundos.
* Speedup (S): mejora del paralelo frente al secuencial  
   S = T\_secuencial / T\_paralelo
* Eficiencia (E): que tan bien se aprovechan los hilos  
   E = (S / P) \* 100, siendo P el numero de hilos.
* Escalabilidad: evolucion de T, S y E al aumentar P.

**Puntos clave / archivos**

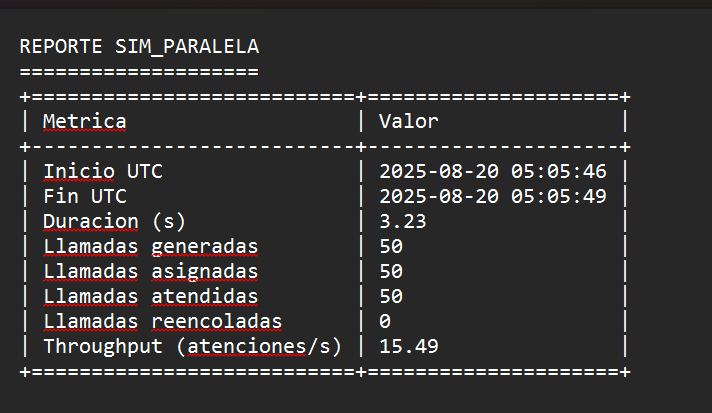
Las tablas de cada .txt ya contienen los valores necesarios para estas fórmulas.

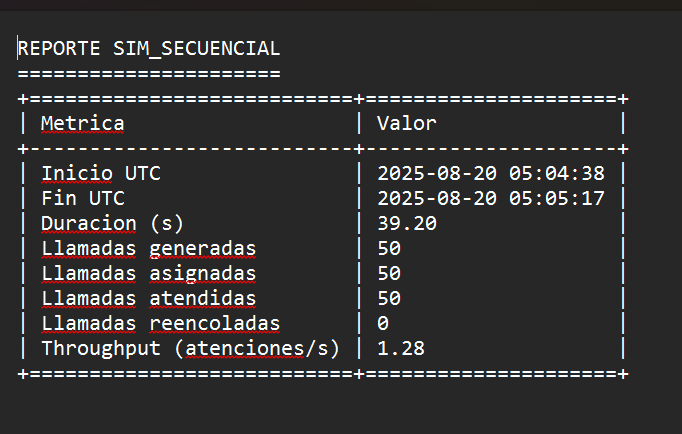
**3. Gráficas o tablas con resultados**

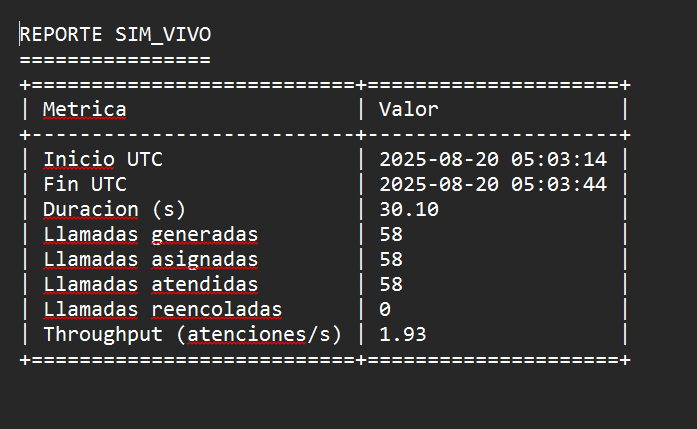
**comparacion\_seq\_vs\_par\_reporte.txt**

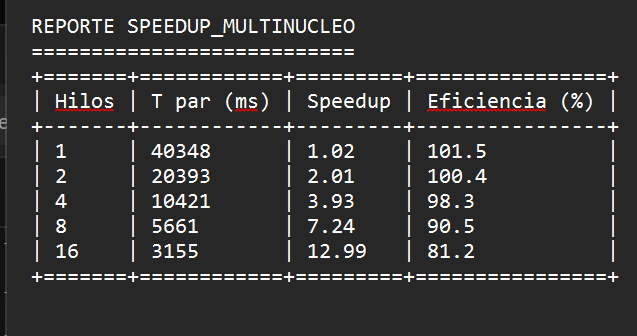
****

**sim\_paralela\_reporte.txt**

****

**sim\_secuencial\_reporte.txt  
  
  
sim\_vivo\_reporte.txt**

**  
  
speedup\_multinucleo\_reporte.txt**

**  
  
4. Análisis de cuellos de botella o limitaciones**

* Granularidad de tareas. Si las unidades son demasiado cortas, el overhead del paralelismo puede comerse la ganancia.
* Contencion en sincronizacion. Uso de BlockingCollection, SemaphoreSlim y secciones con lock puede generar esperas.
* Logs a consola. Escrituras frecuentes y sincronizadas disminuyen el rendimiento y desbalancean hilos.
* Disponibilidad de brigadas. Si pocas brigadas estan disponibles, la cola se procesa mas lento.
* Carga tipo IO. Partes simuladas con Task.Delay limitan speedup; el CPU espera.
* Mejoras sugeridas
* Reducir logs en caliente (acumular y volcar cada N items).
* Minimizar el tamano de las regiones con lock.
* Ajustar la granularidad (tareas un poco mas grandes).
* Evaluar colas por prioridad (si la logica lo permite).
* Probar System.Threading.Channels como alternativa a BlockingCollection.
* Calibrar MaxGradoParalelismo segun mezcla CPU/IO (a veces P < CPU rinde mejor).

**7. Trabajo en Equipo**

1. **Descripción del reparto de tareas**

El equipo estuvo conformado por cinco integrantes, cada uno con un rol específico:

* **Manuel (Líder y Arquitecto del Sistema):** Coordinó el grupo, integró los módulos y definió las clases base e interfaces.
* **Jordi (Especialista en Paralelismo):** Implementó la ejecución en paralelo con TPL, locks, semáforos y colecciones concurrentes.
* **Yerelin (Especialista en Simulación):** Diseñó el generador de emergencias y la lógica de asignación de brigadas.
* **Netanel (Especialista en Datos):** Incorporó datos reales de la República Dominicana y desarrolló el sistema de carga de datos.
* **Ricardo (Especialista en Métricas)**: Encargado de medir rendimiento, comparar ejecución secuencial vs paralela y generar reportes gráficos.

Este reparto permitió trabajar de manera modular y organizada, asegurando que cada miembro se enfocara en una parte clave del proyecto.

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

Para la colaboración y el control de avances se usaron las siguientes herramientas:

**GitHub:** repositorio principal y control de versiones.

**GitHub Desktop:** gestión visual de commits, ramas y merges.

**Microsoft Teams:** reuniones virtuales, comunicación continua y seguimiento del proyecto.

**Google Documents:** redacción colaborativa del informe final.

**WhatsApp:** comunicación rápida y notificaciones al equipo.

## **8. Conclusiones**

### **Principales aprendizajes técnicos**

* Diseño por capas y separación de responsabilidades: La arquitectura modular permitió una clara distribución de funciones, facilitando el mantenimiento, la escalabilidad y la comprensión del sistema.
* Paralelismo efectivo con TPL: El uso de Task.Run y Parallel.ForEach demostró ser eficaz para manejar cargas variables y procesamiento batch, respectivamente. La combinación de patrones productor-consumidor con asincronía permitió una simulación realista y altamente concurrente.
* Sincronización eficiente: El uso de mecanismos como Interlocked y lock en puntos críticos garantizó consistencia sin sacrificar rendimiento. La cancelación cooperativa con CancellationToken permitió una terminación controlada del sistema.
* Evaluación cuantitativa del rendimiento: La implementación de métricas como throughput, speedup y eficiencia permitió una evaluación objetiva del comportamiento del sistema bajo diferentes configuraciones de núcleos, revelando el impacto real del paralelismo.

### **Retos enfrentados y superados**

* Gestión de concurrencia sin bloqueos prolongados: Evitar el uso excesivo de locks fue clave para mantener la fluidez del sistema. Se logró mediante operaciones atómicas y tareas diferidas para liberar recursos.
* Balance de carga entre hilos: La variabilidad en la duración de cada emergencia presentó un desafío para el equilibrio de trabajo. El gestor paralelo extendido permitió recopilar métricas y ajustar configuraciones para mejorar la distribución.
* Simulación realista con datos heterogéneos: Integrar datos reales (JSON de provincias, municipios, brigadas) exigió una lógica robusta para generar emergencias representativas y mantener coherencia en la asignación de brigadas.
* Medición precisa del rendimiento: Diseñar pruebas comparativas entre ejecución secuencial y paralela, y luego escalar a múltiples núcleos, requirió una instrumentación cuidadosa para evitar sesgos y garantizar reproducibilidad.

### **Posibles mejoras o líneas futuras**

* Optimización adaptativa del grado de paralelismo: Implementar un sistema que ajuste dinámicamente el número de hilos según la carga actual podría mejorar aún más la eficiencia.
* Persistencia de métricas históricas: Guardar los resultados de simulaciones anteriores permitiría análisis longitudinales y comparativas más profundas.
* Visualización interactiva de resultados: Integrar gráficos en tiempo real para throughput, uso de CPU y eficiencia facilitaría la interpretación de datos por parte de usuarios no técnicos.
* Simulación distribuida: Escalar el sistema a múltiples nodos o máquinas permitiría evaluar escenarios de emergencia a nivel nacional o regional con mayor realismo.
* Integración con sistemas de alerta reales: Conectarse a fuentes externas de datos (por ejemplo, sensores IoT o redes sociales) permitiría generar emergencias en tiempo real basadas en eventos reales.

**9. Referencias**

**Fuentes bibliográficas, técnicas o académicas consultadas**

* Huracán Fiona (2022) – *Wikipedia en español* ([Wikipedia](https://es.wikipedia.org/wiki/Hurac%C3%A1n_Fiona?utm_source=chatgpt.com))
* Informe de seguimiento y daños – *All Hands and Hearts* ([All Hands and Hearts](https://www.allhandsandhearts.org/es/storm-tracker/hurricane-fiona-2022/?utm_source=chatgpt.com))
* Temporada de huracanes 2022 – *Wikipedia en español* ([Wikipedia](https://es.wikipedia.org/wiki/Temporada_de_huracanes_del_Atl%C3%A1ntico_de_2022?utm_source=chatgpt.com))
* Huracán Beryl (2024) – *Wikipedia en español* ([Wikipedia](https://es.wikipedia.org/wiki/Hurac%C3%A1n_Beryl?utm_source=chatgpt.com))
* Comunicado COE sobre Beryl – *Centro de Operaciones de Emergencias (RD)* ([coe.gob.do](https://www.coe.gob.do/index.php/noticias/item/841-incremento-de-alerta-huracan-beryl-01-de-julio-2024-11-00-pm?utm_source=chatgpt.com))

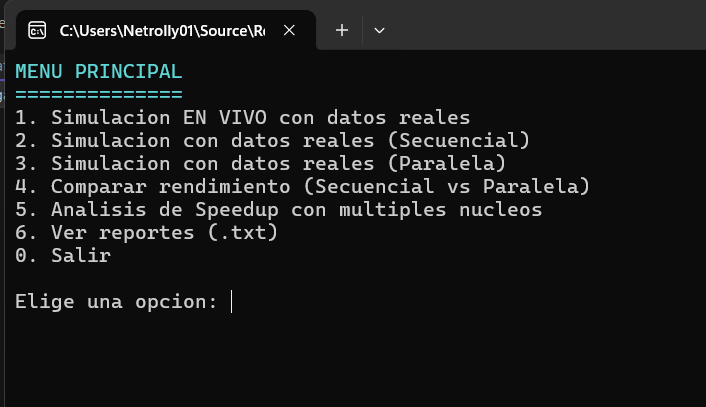
**10. Anexos**

1. **Manual de ejecución del sistema**

### ***1. Abrir el programa***

* Ejecuta el archivo principal (por ejemplo, Program.exe o desde Visual Studio).
* Verás el menú en consola con opciones numeradas del 0 al 6.

### ***2. Elegir una opción del menú***



### ***3. Ver resultados***

* Cada simulación genera un archivo de texto con los resultados.
* Los reportes se guardan en la carpeta: resultados-reportes/
* Puedes verlos con la opción 6 del menú o abrirlos manualmente.

### ***4. Detener simulación en vivo (opción 1)***

* Si estás en la simulación en vivo, puedes detenerla con una tecla (según implementación) o cerrando la consola.
* El sistema usa cancelación cooperativa para terminar sin errores.

### ***5. Repetir o cambiar simulación***

* Vuelve al menú principal después de cada ejecución.
* Puedes probar diferentes opciones para comparar resultados.

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

<https://github.com/Manushark/BrigadasEmergenciaRD>