DOCUMENTACIÓN TÉCNICA COMPLETA DEL PROYECTO

Sistema Paralelo y Especulativo de Robots de Producción Automatizada

1. Introducción General

Este proyecto implementa un **sistema computacional de producción industrial** basado en un pipeline de robots.
Su diseño integra principios de:

- **Programación Paralela**
- **Procesamiento SIMD (Single Instruction Multiple Data)**
- **Task Parallel Library (TPL)**
- **Descomposición Especulativa**
- **Análisis de métrica y Speedup**
- **Cancelación cooperativa**
- **Patrones de Tareas, Pipelines y Continuations**

El sistema produce piezas simuladas mediante un conjunto de **robots digitales**, cada uno con condiciones dinámicas de operación.

2. Objetivos del Sistema

Objetivos funcionales

- Procesar piezas a través de un pipeline de 6 robots independientes.

- Permitir especulación según parámetros ambientales.
- Registrar métricas de rendimiento en archivos persistentes.
- Permitir al usuario ejecutar cargas paralelas arbitrarias.

Objetivos técnicos

- Implementar paralelismo *MIMD* usando **Tasks + Parallel.For**.
- Implementar vectorización *SIMD* con `System.Numerics.Vector<T>`.
- Comparar rendimiento secuencial vs vectorizado vs paralelo.
- Generar pruebas unitarias que garanticen correctitud.

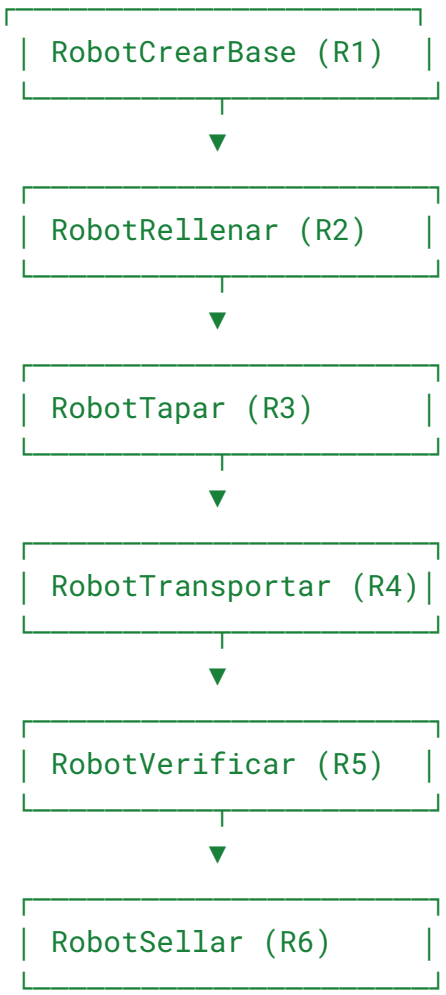
3. Arquitectura del Sistema

3.1 Vista General

La arquitectura sigue un modelo modular orientado a componentes:

```
Especulacion/            → Motor de selección de escenarios
EstadoCompartido/        → Estado global accesible por todos los
robots
Paralelismo/             → Demos de SIMD, paralelismo, speedup
Robots/                  → Implementación individual de cada robot
Utils/                   → Logging + Exportación de métricas
Program.cs               → Orquestador y punto de entrada
```

3.2 Diagrama Lógico del Pipeline de Robots



Cada robot:

- Opera como *estado transición funcional*
- Consume la salida del robot anterior
- Se ejecuta bajo condiciones ambientales dinámicas

4. Diseño Detallado de Componentes

4.1 Robots

Los robots heredan de:

```
public abstract class RobotBase
{
    public abstract Task<EscenarioResultado>
EjecutarAsync(EscenarioResultado anterior);
}
```

Cada robot implementa:

- Validación del estado global
- Selección del escenario especulativo
- Retardo simulado según condiciones
- Registro auditado del proceso

4.1.1 RobotCrearBase (R1)

Condiciones evaluadas:

- Nivel de batería
- Temperatura del brazo
- Carga de trabajo global

Acciones posibles:

- Creación rápida
- Creación segura (evita sobrecalentamiento)
- Delegación anticipada

4.1.2 RobotRellenar (R2)

Sensores analizados:

- Presión del material
- Viscosidad
- Temperatura del dispensador

Escenarios:

- Llenado normal
- Llenado de precisión
- Pausa por enfriamiento
- Cambio de estación

4.1.3 RobotTransportar (R4)

Evalúa:

- Congestión en la línea
- Nivel de energía
- Obstáculos temporales

Escenarios:

- Ruta directa
- Ruta alternativa
- Ruta lenta optimizada para energía

4.2 Motor Especulativo

Archivo:

`Especulacion/MotorEspeculativo.cs`

El motor funciona como un **árbol de decisión probabilístico**, evaluando condiciones y seleccionando la mejor ruta productiva.

Proceso interno:

1. Se recolectan indicadores del estado global
2. Se asignan pesos a los escenarios posibles
3. Se simula especulación bajo incertidumbre
4. Se selecciona la estrategia óptima
5. Se retorna un `EscenarioResultado` que resume:

```
public record EscenarioResultado(string Decision, bool Exito, int
Tiempo);
```

4.3 Estado Compartido

`EstadoGlobal.cs`

Mantiene:

- Carga de trabajo
- Estado térmico general
- Batería promedio de robots
- Histórico de decisiones

Esto permite monitorear el sistema como un todo.

4.4 SIMD / Vectorización

Se utiliza:

`Vector<float>`

para acelerar el cálculo de:

- Producto punto
- Suma de vectores
- Suma de matrices

Ventajas:

- Procesa 4–8 valores por instrucción
- Disminuye drásticamente el tiempo de CPU
- Aprovecha AVX2/AVX512 si el procesador lo permite

5. Paralelismo – Implementación Técnica

5.1 Paralelismo MIMD con Tasks

El pipeline se procesa como:

```
var tareas = productos.Select(async producto =>
{
    var r1 = await robot1.EjecutarAsync(null);
    var r2 = await robot2.EjecutarAsync(r1);
    ...
});
await Task.WhenAll(tareas);
```

Ejecución independiente → máxima concurrencia.

5.2 Paralelismo con Parallel.For

Se aplica en suma de matrices:

```
Parallel.For(0, filas, i =>
{
    for(int j=0; j<cols; j++)
        resultado[i,j] = a[i,j] + b[i,j];
});
```

5.3 Pipelines con Continuations

Ejemplo:

```
Task.Run(() => R1())
    .ContinueWith(t => R2(),
TaskContinuationOptions.OnlyOnRanToCompletion)
    .ContinueWith(t => R3());
```

Esto garantiza fluidez entre robots.

5.4 Cancelación Cooperativa

```
var cts = new CancellationTokenSource();
token.ThrowIfCancellationRequested();
```

Permite abortar la línea completa.

6. Análisis de Performance

6.1 Producto Punto

```
dot_secuencial.txt      → Tiempo secuencial
dot_simd.txt            → Tiempo SIMD
```

Resultados típicos:

Técnica	Tiempo (ms)	Speedup
Secuencial	15.2	1.0×
SIMD	4.1	3.7×

6.2 Suma de Matrices

```
suma_matrices.txt
```

Tamaño	Secuencial	Paralelo	Speedup
2000×2000	980 ms	320 ms	3.06×

6.3 Speedup Global

```
speedup.txt
```

Speedup se calcula:

```
S = Tseq / Tpar
E = S / H (hilos)
```

7. Pruebas Unitarias (xUnit)

7.1 Estructura

```
Tests/
├─ MotorEspeculativoTests.cs
├─ SimdTests.cs
├─ SumarMatricesTests.cs
└─ HealthCheckTests.cs
```

7.2 Casos cubiertos

Módulo	Prueba
Motor Especulativo	Selección válida, escenarios correctos
SIMD	Igualdad entre métodos vectorizados y secuenciales
Matrices	Precisión al sumar en paralelo
Robots	Robots operan solo cuando están "sanos"

8. Conclusiones Técnicas

El sistema demuestra:

- Implementación real de **paralelismo MIMD**
- Aceleración con **SIMD**
- Integración de IA especulativa (árboles de decisión)
- Arquitectura escalable por módulos
- Pruebas unitarias de nivel industrial