Informe: Estrategia de Paralelización del Mini Mini Proyecto

Estudiante: Diederich Solis 22952
Repositorio: `https://github.com/DiederichSolis/proyecto_trafico.git`

1. Descripción del Problema

Este proyecto simula el tráfico en una intersección con varios semáforos y vehículos. Cada vehículo se asocia a un semáforo y su movimiento depende del estado de éste:

- **VERDE**: avanza a su velocidad asignada.
- **AMARILLO**: avanza un metro por iteración.
- **ROJO**: se detiene.

Los semáforos cambian de estado según tiempos predefinidos. La simulación se ejecuta en múltiples iteraciones, mostrando el estado de cada semáforo y la posición de los vehículos.

2. Estrategia de Paralelización

Se identificaron dos tareas independientes y costosas que se repiten en cada iteración:

1. Actualizar el estado de todos los semáforos.
2. Mover todos los vehículos según su semáforo.

Estas tareas se paralelizaron de dos maneras:

- **Paralelización de bucles**: con `#pragma omp parallel for` para dividir el trabajo de actualización de semáforos y movimiento de vehículos entre múltiples hilos.
- **Paralelización por secciones**: con `#pragma omp parallel sections` para que la actualización de semáforos y el movimiento de vehículos ocurran en paralelo, cada uno en su propia sección.

3. Justificación del uso de OpenMP

OpenMP es una API estándar para paralelismo en memoria compartida que:

- Permite paralelizar rápidamente código secuencial mediante directivas.
- Soporta **paralelismo dinámico** (`omp_set_dynamic`) para ajustar el número de hilos en tiempo de ejecución.
- Facilita la **planificación** (`schedule(static)` y `schedule(dynamic,chunk)`) para balancear la carga de trabajo.
- Es portable y no requiere cambios drásticos en la estructura del código.

4. Código Secuencial

La versión secuencial (`simulacion_secuencial.c`) ejecuta la simulación en un solo hilo, actualizando secuencialmente los semáforos y luego moviendo los vehículos.

Fragmento de `simulacion_secuencial.c`

```
void simulate_traffic(int iterations, Vehicle *v, int nveh,
                      TrafficLight *L, int nlights, int print_every) {
    for (int t = 1; t <= iterations; t++) {
        update_traffic_lights(L, nlights);
        move_vehicles(v, nveh, L);
        ...
    }
}
```

5. Código Paralelo

La versión paralela (`simulacion_paralela.c`) incorpora directivas OpenMP para acelerar el procesamiento.

Fragmentos destacados de `simulacion_paralela.c`

```
#pragma omp parallel for schedule(static)
for (int i = 0; i < n; i++) { ... } // Actualizaci n de sem foros

#pragma omp parallel for schedule(dynamic, 8)
for (int i = 0; i < nveh; i++) { ... } // Movimiento de veh culos

#pragma omp parallel sections
{
    #pragma omp section
    update_traffic_lights(L, nlights);
    #pragma omp section
    move_vehicles(v, nveh, L, nlights);
}
```

6. Conclusiones

El uso de OpenMP permitió:

- Reducir el tiempo de ejecución al distribuir el trabajo entre múltiples hilos.
- Aprovechar la independencia entre la actualización de semáforos y el movimiento de vehículos.
- Implementar de forma sencilla el paralelismo con cambios mínimos sobre el código secuencial original.