Mini-Proyecto - Tráfico

Nombre del estudiante:

Sergio Orellana - 221122

Curso:

Computación Paralela y Distribuida

13 de agosto de 2025

1. Breve descripción del problema

El proyecto consiste en simular tráfico vehicular en una intersección con múltiples semáforos. Cada vehículo obedece a un semáforo específico y decide si avanza o se detiene según el *estado* del semáforo (rojo, verde o amarillo). En cada iteración de la simulación se realizan dos pasos principales:

a) **Actualizar semáforos**: decrementar temporizadores y realizar la transición de estados (`GREEN` → `YELLOW` → `RED` → `GREEN`).

b) **Mover vehículos**: calcular si cada vehículo avanza y, de ser así, actualizar su posición con base en su velocidad.

La versión secuencial ejecuta estas operaciones en un único hilo. La versión paralela busca disminuir el tiempo de ejecución utilizando `OpenMP` para distribuir el trabajo en varios hilos.

2. Estrategia de paralelización utilizada

La paralelización se diseñó en dos niveles complementarios:

1. Paralelismo por *tareas* (secciones)

Se ejecutan en paralelo las dos tareas de alto nivel por iteración:

- **Actualización de semáforos**
- **Movimiento de vehículos**

Esto se implementa con `#pragma omp parallel sections`, creando secciones independientes que pueden correr simultáneamente:

```
#pragma omp parallel sections
{
  #pragma omp section
  { update_traffic_lights_parallel(lights, num_lights); }

  #pragma omp section
  { move_vehicles_parallel(vehicles, num_vehicles, lights, num_lights);
     }
}
```

Listing 1: Paralelismo por secciones

2. Paralelismo de *datos* dentro de cada tarea

Cada sección interna también se paraleliza:

- **Semáforos**: un bucle `for` paralelo con `schedule(static)` distribuye uniformemente los semáforos entre hilos, pues el costo por semáforo es similar.
- **Vehículos**: un bucle `for` paralelo con `schedule(dynamic, 4)` asigna *chunks* pequeños y dinámicos a los hilos, equilibrando carga cuando hay divergencias (p.ej. diferentes velocidades o decisiones de avance).

```
// Sem foros: costo homog neo
#pragma omp parallel for schedule(static)
for (int i = 0; i < num_lights; ++i) { /* actualizar timer/estado */ }

// Veh culos: costo potencialmente heterog neo
#pragma omp parallel for schedule(dynamic, 4)
for (int i = 0; i < num_vehicles; ++i) { /* decidir avance y mover */ }
```

Listing 2: Bucle de semáforos (estático) y vehículos (dinámico)

3. Ajuste dinámico de hilos y paralelismo anidado

Para adaptarse al tamaño de la entrada, en cada iteración se ajusta el número de hilos según la cantidad de vehículos y se habilita paralelismo anidado:

```
omp_set_dynamic(1);       // permitir variaci n de hilos por el runtime
omp_set_nested(1);        // permitir paralelismo anidado (for dentro de
    sections)

int num_threads = num_vehicles / 10 + 2;
if (num_threads < 2) num_threads = 2;
omp_set_num_threads(num_threads);
```

Listing 3: Ajuste dinámico y paralelismo anidado

3. Justificación del uso de OpenMP

- **Modelo simple y portable**: `OpenMP` permite paralelizar gradualmente código C existente con directivas (`#pragma`) sin rediseñar por completo la arquitectura.
- **Paralelismo híbrido (tareas + datos)**: con `sections` se explota paralelismo *entre* tareas (actualizar semáforos vs. mover vehículos) y con `parallel for` se explota paralelismo *dentro* de cada tarea.

- **Balanceo de carga con `schedule(dynamic)`**: el movimiento de vehículos puede tener costo desigual; el *scheduling* dinámico asigna trabajo de forma adaptativa, evitando hilos ociosos.
- **Paralelismo anidado**: habilitar `omp_set_nested(1)` permite tener un `for` paralelo dentro de `sections`, maximizando el uso de núcleos cuando hay suficiente trabajo en ambas tareas.
- **Ajuste dinámico de hilos**: `omp_set_dynamic(1)` y el cálculo de `num_threads` por iteración permiten adaptarse a entradas pequeñas o grandes sin recompilar, mejorando el rendimiento global.
- **Sobrecarga baja y mantenimiento**: para este tipo de simulación discreta, OpenMP introduce menos complejidad que alternativas como paso de mensajes, y mantiene el código cercano a la versión secuencial.

Referencias

[1] GeeksforGeeks. (2025c, julio 11). *OpenMP | Introduction with Installation Guide.* https://www.geeksforgeeks.org/installation-guide/openmp-introduction-with-installation-guide/.

[2] GeeksforGeeks. (2025c, julio 11). *OpenMP | Hello World program.* https://www.geeksforgeeks.org/c/openmp-hello-world-program/.

[3] GeeksforGeeks. (2023, 19 marzo). *Introduction to Parallel Programming with OpenMP in C++.* https://www.geeksforgeeks.org/cpp/introduction-to-parallel-programming-with-openmp-in-cpp/.

[4] GeeksforGeeks. (2025g, agosto 6). *Implementation of Quick sort using MPI, OMP and Posix thread.* https://www.geeksforgeeks.org/dsa/implementation-of-quick-sort-using-mpi-omp-and-posix-thread/.

[5] GeeksforGeeks. (2025f, julio 23). *Estimating the value of Pi using Monte Carlo | Parallel Computing Method.* https://www.geeksforgeeks.org/dsa/estimating-the-value-of-pi-using-monte-carlo-parallel-computing-method/.

[6] GeeksforGeeks. (2025g, julio 23). *Producer Consumer Problem in C.* https://www.geeksforgeeks.org/c/producer-consumer-problem-in-c/.

[7] GeeksforGeeks. (2025g, julio 23). *Parallel Dijkstra's Algorithm: SSSP in Parallel.* https://www.geeksforgeeks.org/dsa/parallel-dijkstras-algorithm-sssp-in-parallel/?utm_source=chatgpt.com..