МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

ІНСТИТУТ КОМП’ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

кафедра систем штучного інтелекту

ЗВІТ

про виконання курсової роботи

з курсу «Штучний інтелект в ігрових застосунках»
на тему «Паралельна обробка потоків даних у системах керування дорожнім рухом»

Виконав:

ст. групи КНСШ-11

Чорній Юрій Юрійович

Перевірила:

Гентош Л. І.

ЛЬВІВ – 2023

Зміст

Вступ

Системи керування дорожнім рухом відіграють життєво важливу роль в управлінні та оптимізації транспортних потоків у містах. Ефективна обробка потоків даних, що генеруються різними датчиками та пристроями, має вирішальне значення для ефективного прийняття рішень та управління в реальному часі в цих системах. Це дослідження фокусується на паралельній обробці потоків даних в системах управління дорожнім рухом з метою підвищення швидкості і масштабованості процесів аналізу даних і прийняття рішень. Використовуючи можливості паралельних обчислень, це дослідження спрямоване на вирішення зростаючих проблем, пов'язаних зі збільшенням інтенсивності руху, заторами та необхідністю реагування в реальному часі при управлінні міським рухом. Актуальність та новизна цього дослідження полягає в його потенціалі для підвищення ефективності та результативності систем управління дорожнім рухом, що призведе до покращення транспортного потоку, зменшення заторів та підвищення транспортної стійкості.

Згідно з [1] фреймворк SUMO (Simulation of Urban MObility) привернув значну увагу до моделювання дорожнього руху, надаючи реалістичну і гнучку платформу для моделювання та аналізу транспортних потоків у містах. В статті [4] підкреслили важливість паралельного виконання у великомасштабному агентному моделюванні дорожнього руху, що забезпечує швидші обчислення та масштабованість. Методи паралельної обробки можуть ефективно розподіляти обчислювальне навантаження між декількома обчислювальними одиницями, такими як CPU або GPU. Робота [5] продемонструвала потенціал паралельного моделювання на GPU для великомасштабних міських транспортних потоків. Використовуючи обчислювальну потужність графічних процесорів, можна значно підвищити продуктивність моделювання, що дозволяє обробляти більш детальні та складні моделі руху в режимі реального часу.

В [6] запропонували алгоритм паралельного моделювання великомасштабних міських транспортних мереж з використанням графічних процесорів. Дослідження підкреслило переваги розпаралелювання на GPU з точки зору прискорення та обчислювальної ефективності порівняно з традиційними моделюваннями на базі CPU. Згідно з [7] дослідили методи прискорення на базі GPU для великомасштабних симуляцій дорожнього руху. Їхнє дослідження показало, що розпаралелювання на GPU може значно скоротити час симуляції та дозволити працювати з більш розгалуженими транспортними мережами. У [8] представили підхід на основі GPU для паралельного моделювання міського трафіку, продемонструвавши його потенціал для контролю та управління трафіком у реальному часі. Дослідження продемонструвало переваги розпаралелювання на GPU з точки зору швидкості та масштабованості.

В [11] зосередилися на паралельному моделюванні автомобільного руху на базі GPU з динамічною системою навігації по маршруту. Дослідження підкреслило потенціал розпаралелювання на GPU для прийняття рішень у реальному часі в системах керування дорожнім рухом. [12] запропонували підхід до мікросимуляції на основі CUDA для моделювання міської транспортної мережі з використанням розпаралелювання на GPU. Дослідження продемонструвало ефективність та результативність розпаралелювання на базі GPU для великомасштабних симуляцій дорожнього руху. В статті [13] розробили алгоритм планування траєкторії руху транспортних засобів у реальному часі з прискоренням на GPU для симуляції міського руху. Дослідження підкреслило переваги розпаралелювання на графічних процесорах у забезпеченні контролю та оптимізації руху в реальному часі.

Метою цього дослідження є вивчення застосування методів паралельної обробки, зокрема розпаралелювання графічних процесорів, в аналізі потоків даних для систем керування дорожнім рухом. Дослідження спрямоване на оцінку продуктивності та масштабованості підходів паралельної обробки та їх впливу на прийняття рішень в реальному часі в системах керування дорожнім рухом.

Було висунуто наступні гіпотези:

Гіпотеза 1: Розпаралелювання на графічних процесорах може значно підвищити швидкість обробки та масштабованість потоків даних у системах керування дорожнім рухом, що дозволить приймати рішення в реальному часі. Ця гіпотеза пов'язана з перевагами, продемонстрованими в літературі щодо прискорення, яке досягається завдяки розпаралелюванню графічних процесорів (Rupprecht та ін., 2019 [4]; Chen, M., & Chen, J, 2017 [9]).

Гіпотеза 2: Методи паралельної обробки, включаючи розпаралелювання як CPU, так і GPU, можуть підвищити ефективність і результативність систем управління дорожнім рухом, дозволяючи проводити аналіз і приймати рішення в реальному часі. Ця гіпотеза підтверджується результатами досліджень, які демонструють переваги розпаралелювання у скороченні часу моделювання та роботі з більшими транспортними мережами (Lu and Aboudolas, 2018 [7]; Belalem and Bougourzi, 2020 [8]).

Постановка проблеми полягає в тому, що традиційні методи обробки даних в системах управління дорожнім рухом можуть не справлятися зі зростаючим обсягом і складністю потоків даних, що призводить до затримок у прийнятті рішень і неоптимального управління дорожнім рухом. Тому існує потреба у дослідженні підходів до паралельної обробки, зокрема розпаралелювання на графічних процесорах, для подолання цих викликів та підвищення ефективності систем керування дорожнім рухом.

Аналіз літературних джерел

Сучасний стан паралельної обробки потоків даних у системах керування дорожнім рухом характеризується значними досягненнями та багатообіцяючими результатами досліджень. Дослідники визнали важливість ефективної обробки даних і прийняття рішень в реальному часі в управлінні дорожнім рухом, що призвело до появи великої кількості літератури, в якій досліджуються методи паралельних обчислень, зокрема, розпаралелювання на графічних процесорах, в цій галузі.

Переваги паралельних обчислень в системах керування дорожнім рухом очевидні з існуючих досліджень. По-перше, методи паралельної обробки можуть значно підвищити швидкість обробки і масштабованість потоків даних. Це дозволяє проводити аналіз і приймати рішення в реальному часі, що має вирішальне значення для ефективного управління дорожнім рухом в динамічному міському середовищі.

Ще однією перевагою є потенціал для підвищення обчислювальної ефективності. Паралельна обробка розподіляє обчислювальне навантаження між кількома обчислювальними блоками, що дозволяє ефективно використовувати ресурси і швидше виконувати алгоритми. Це призводить до підвищення продуктивності та можливості обробляти більш детальні та складні моделі трафіку в режимі реального часу.

Хоча існуючі дослідження паралельної обробки в системах керування дорожнім рухом дали позитивні результати, є деякі недоліки, які потребують вирішення. Одним з обмежень є відсутність комплексних досліджень, які вивчають практичну реалізацію та інтеграцію методів паралельної обробки в реальні системи управління дорожнім рухом. Багато досліджень зосереджені на аспектах симуляції та моделювання, але існує потреба в додаткових дослідженнях практичної застосовності та інтеграції паралельних обчислень в операційні системи управління дорожнім рухом.

Крім того, масштабованість підходів до паралельної обробки потребує подальшого вивчення. Хоча розпаралелювання продемонструвало переваги в роботі з великими транспортними мережами, існує потреба дослідити його продуктивність при роботі з дуже перевантаженими і складними міськими районами з величезним потоком даних. Крім того, необхідно ретельно оцінити компроміс між обчислювальною складністю та точністю, оскільки більша розпаралеленість може призвести до втрати точності в певних сценаріях.

Запропонований підхід має на меті кількісно оцінити переваги та потенційні компроміси паралельної обробки з точки зору обчислювальної складності та точності. Ретельно оцінюючи обчислювальні витрати та ефективність, це дослідження прагне забезпечити комплексне розуміння практичних переваг та обмежень паралельної обробки в системах керування дорожнім рухом.

Матеріали та методи

Опис алгоритму для розпаралелювання даних отриманих із симуляції SUMO:

1. Ініціалізація середовища моделювання:

- ❖ Завантаження дорожньої мережі та даних про транспортні засоби.
- ❖ Ініціалізація параметрів симуляції, такі як час симуляції, часовий крок та налаштування графічного процесора.

```
import sumolib
import traci
import numpy as np
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule

# Load SUMO network and start simulation
sumo_cmd = ["sumo", "-c", "simple.sumocfg"]
traci.start(sumo_cmd)
traci.simulationStep(100)
# Get the number of vehicles in the simulation
num_vehicles = traci.vehicle.getIDCount()

# Initialize positions on the CPU
positions_cpu = np.zeros((num_vehicles, 2), dtype=np.float32)

# Create GPU memory for positions
positions_gpu = cuda.mem_alloc(positions_cpu.nbytes)

# CUDA kernel for updating vehicle positions
cuda_kernel = """
__global__ void update_positions(float2* positions, int num_vehicles) {
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    if (tid < num_vehicles) {
        positions[tid].x = traci.vehicle.getPosition(tid)[0];
        positions[tid].y = traci.vehicle.getPosition(tid)[1];
    }
}
"""
```

Рис.1 Ініціалізація середовища моделювання

2. Генерація початкових положеннь та швидкості транспортних засобів:

- ❖ Розподілення транспортних засобів по дорожній мережі.
- ❖ Призначення початкових швидкостей та прискорення для кожного транспортного засобу.

```
<configuration>
    <input>
        <net-file value="map.net.xml"/>
        <route-files value="truck_routes.rou.xml, passenger_routes.rou.xml,
        <additional-files value="map.poly.xml"/>
    </input>

    <time>
        <begin value="0"/>
        <end value="10000"/>
    </time>


    <gui_only>
        <gui-settings>
            <scale value="1"/>
            <delay value="150"/>
        </gui-settings>
    </gui_only>
</configuration>
```

Рис.2 Конфігураційний файл симуляції

```
    <vehicle id="bus4" type="bus" depart="120.00">
        <route edges="183555573#0 183555573#1 1835555
    </vehicle>
```

Рис.3 Приклад з файлу генерації транспортних засобів

```
<trip id="0" depart="0.00" from="352399687" to="-158031379#1"/>
<trip id="1" depart="0.50" from="139932906" to="-965353311"/>
<trip id="2" depart="1.00" from="41461196#3" to="391341941#0"/>
<trip id="3" depart="1.50" from="641491652#0" to="-24567417#2"/>
<trip id="4" depart="2.00" from="183555573#0" to="-158551373#4"/>
<trip id="5" depart="2.50" from="-661300188" to="24567355#0"/>
<trip id="6" depart="3.00" from="-158031379#2" to="-641236159#1"/>
<trip id="7" depart="3.50" from="353982926" to="253498831"/>
<trip id="8" depart="4.00" from="-696378835#2" to="32847601#3"/>
<trip id="9" depart="4.50" from="47932115#0" to="-1154607033#1"/>
<trip id="10" depart="5.00" from="4774858#1" to="39763728#4"/>
```

Рис.4 Призначення початкової та кінцевої цілі

3. Ініціалізація пам'яті GPU:

- Виділення пам'яті на графічному процесорі для зберігання положень транспортних засобів, швидкостей та інших даних симуляції.
- Копіювання початкових даних про транспортні засоби з CPU в пам'ять GPU.

```
# Get the number of vehicles in the simulation
num_vehicles = traci.vehicle.getIDCount()

# Initialize positions on the CPU
positions_cpu = np.zeros((num_vehicles, 2), dtype=np.float32)

# Create GPU memory for positions
positions_gpu = cuda.mem_alloc(positions_cpu.nbytes)
```

Рис.5 Ініціалізація пам'яті GPU

4. Цикл симуляції:

- Поки час симуляції знаходиться в межах бажаного діапазону:
  - Виконати наступні кроки паралельно на GPU:
    - Оновлення положень та швидкостей транспортних засобів на основі прискорення та поточних швидкостей.
    - Перевірка наявності зіткнень та застосовувати відповідні стратегії вирішення зіткнень.
  - Оновлювати стани світлофорів та обробляйти логіку керування дорожнім рухом.
- Синхронізувати потоки GPU для забезпечення узгодженості даних.

```
# Simulation loop
while traci.simulation.getMinExpectedNumber() > 0:
    # Get the updated number of vehicles
    num_vehicles = traci.vehicle.getIDCount()

    # Resize GPU memory if the number of vehicles has changed
    if num_vehicles != positions_cpu.shape[0]:
        # Free existing GPU memory
        positions_gpu.free()

        # Allocate new GPU memory for positions
        positions_cpu = np.zeros((num_vehicles, 2), dtype=np.float32)
        positions_gpu = cuda.mem_alloc(positions_cpu.nbytes)

    # Update vehicle positions using GPU parallelism
    update_positions_gpu(positions_gpu, np.int32(num_vehicles), block=(block_size, 1, 1))

    # Copy updated positions from GPU to CPU
    cuda.memcpy_dtoh(positions_cpu, positions_gpu)

    # Advance the simulation
    traci.simulationStep()
```

Рис.6 Цикл симуляції

5. Отримання результатів симуляції:

- ❖ Копіювання кінцевих положень, швидкостей та інших важливих даних з пам'яті графічного процесора назад до центрального процесора.

```
# Copy updated positions from GPU to CPU
cuda.memcpy_dtoh(positions_cpu, positions_gpu)
```

Рис.7 Копіювання результатів з GPU до CPU

```
# Run performance analysis for each approach
cpu_memory_1, cpu_time_1 = measure_performance(cpu_parallel(1))
cpu_memory_2, cpu_time_2 = measure_performance(cpu_parallel(2))
cpu_memory_4, cpu_time_4 = measure_performance(cpu_parallel(4))
gpu_memory_128, gpu_time_128 = measure_performance(gpu_parallel(128))
gpu_memory_256, gpu_time_256 = measure_performance(gpu_parallel(256))
gpu_memory_512, gpu_time_512 = measure_performance(gpu_parallel(512))

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(cpu_time_1, cpu_memory_1, label='Non-Parallel CPU')
plt.plot(cpu_time_2, cpu_memory_2, label='Parallel CPU (2 cores)')
plt.plot(cpu_time_4, cpu_memory_4, label='Parallel CPU (4 cores)')
plt.plot(gpu_time_128, gpu_memory_128, label='GPU (128 threads)')
plt.plot(gpu_time_256, gpu_memory_256, label='GPU (256 threads)')
plt.plot(gpu_time_512, gpu_memory_512, label='GPU (512 threads)')
plt.xlabel('Execution Time (s)')
plt.ylabel('Memory Usage (bytes)')
plt.title('Performance Analysis')
plt.legend()
plt.show()
```

Рис.8 Зображення аналізу розпаралелювання

Вибір технології паралельних обчислень, такої як розпаралелювання на графічному процесорі, обґрунтований його здатністю обробляти завдання з високим ступенем розпаралелювання та ефективно обробляти великі об'єми даних. Графічні процесори мають численні паралельні обчислювальні блоки (ядра CUDA), які можуть виконувати кілька потоків одночасно, що робить їх добре придатними для симуляції дорожнього руху з великою кількістю транспортних засобів і складними взаємодіями.

Для розробки алгоритму паралельних обчислень були використані програмні фреймворки, такі як CUDA (Compute Unified Device Architecture). CUDA надає модель програмування та інструменти для програмування на GPU, що дозволяє ефективно використовувати обчислювальну потужність графічних процесорів. Вона

надає функції та бібліотеки для керування пам'яттю, синхронізації потоків та паралельного виконання.

Обчислювальна складність запропонованого алгоритму залежить від таких факторів, як розмір дорожньої мережі, кількість транспортних засобів та складність моделі імітації дорожнього руху. Позначимо N - кількість транспортних засобів, а M - кількість часових кроків симуляції.

Розпаралелювання циклу моделювання на графічному процесорі зменшує часову складність з O(N*M) в однопотоковій реалізації на CPU до O(N) або O(M) в реалізації на GPU. Таке зменшення досягається за рахунок одночасного виконання декількох кроків моделювання на різних потоках графічного процесора.

Кількісну оцінку скорочення можна виміряти, порівнюючи час виконання симуляції, розпаралеленої на GPU, з симуляцією на CPU для одного і того ж вхідного сценарію.

Важливо зазначити, що фактичне скорочення може відрізнятися залежно від конкретного сценарію моделювання, апаратного забезпечення GPU та деталей реалізації. Профілювання продуктивності та бенчмаркінг можуть надати більш точні оцінки зменшення обчислювальної складності та прискорення, досягнутого завдяки розпаралелюванню на GPU.

Результати досліджень

Симуляція SUMO (Simulation of Urban MObility) надає різні вхідні дані, які відображають різні аспекти поведінки та динаміки дорожнього руху. Ці дані необхідні для аналізу та моделювання транспортних потоків у міському середовищі. Ось деякі з ключових вхідних даних, які можуть бути отримані в результаті моделювання:

1. Дані про дорожню мережу:

- Топологія доріг: Інформація про розташування доріг, перехресть та сполучення між ними.
- Конфігурації смуг руху: Детальна інформація про кількість смуг руху, ширину смуг та властивості смуг, такі як обмеження швидкості та типи смуг (наприклад, звичайні, поворотні, автобусні смуги).
- Розклад світлофорів: Розклади та схеми роботи світлофорів на перехрестях.

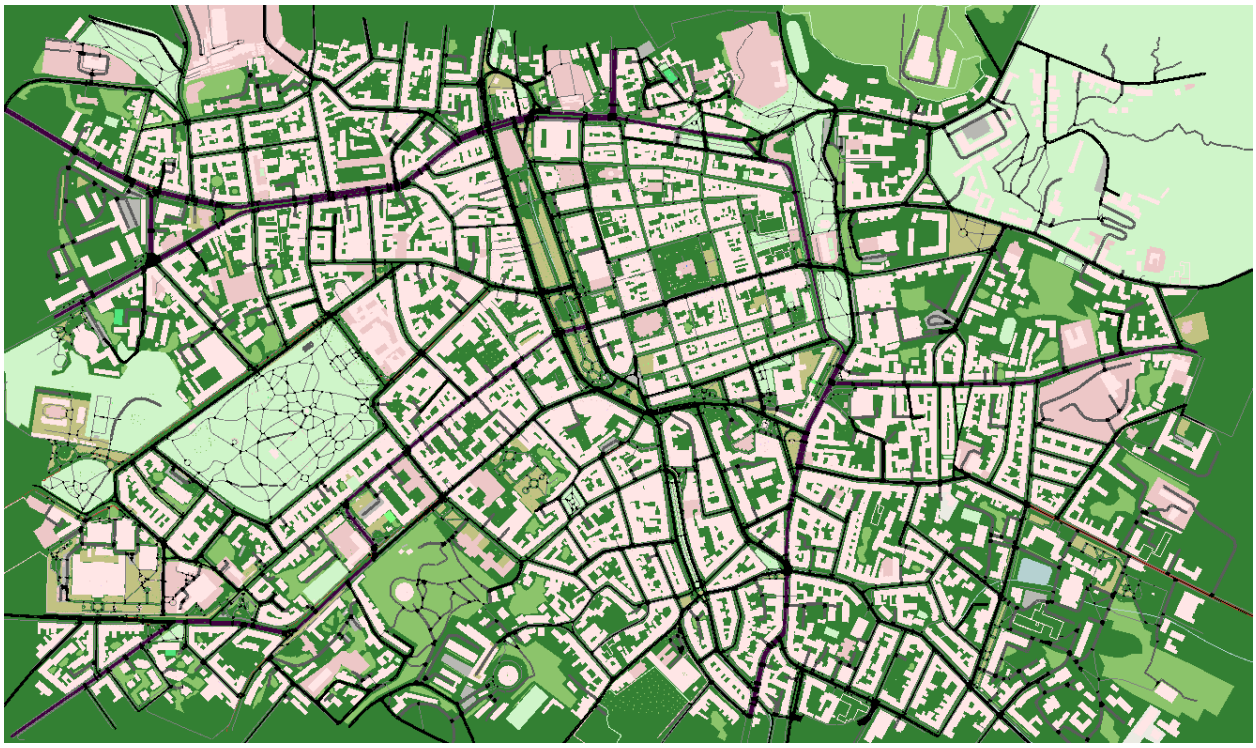

Рис.9 Топологія доріг

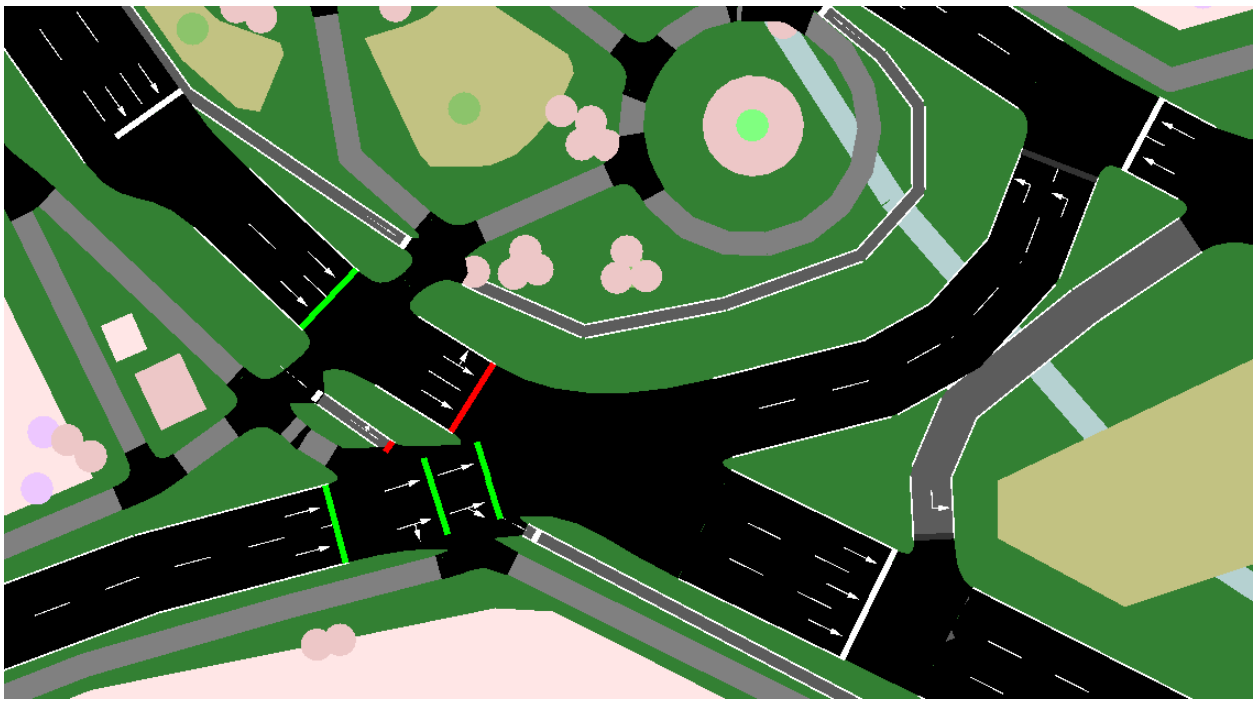

Рис.10 Розклад світлофорів

2. Дані про транспортні засоби:

- ❖ Траєкторії руху транспортних засобів: Положення та швидкість окремих транспортних засобів у часі, що фіксує їх рух протягом симуляції.
- ❖ Атрибути транспортних засобів: Інформація про транспортні засоби, така як тип (легковий автомобіль, вантажівка, автобус), розмір, максимальна швидкість і здатність до прискорення.
- ❖ Маршрути транспортних засобів: Заплановані маршрути руху транспортних засобів, включаючи пункти відправлення та призначення.

Рис.11 Траєкторія руху транспортних засобів

Рис.12 Маршрути транспортних засобів

3. Дані про транспортні потоки:

- ❖ Обсяги руху: Кількість транспортних засобів, що проїжджають через певні пункти або ділянки дорожньої мережі протягом певного періоду часу.
- ❖ Щільність руху: Кількість транспортних засобів на одиницю довжини або площі, що дає уявлення про рівень заторів.
- ❖ Швидкість руху: Середня швидкість або миттєва швидкість транспортних засобів у різних місцях або часових інтервалах.
- ❖ Час у дорозі: Час, який транспортні засоби витрачають на поїздку між конкретними пунктами відправлення та призначення.

Рис.13 Дані про транспортні потоки

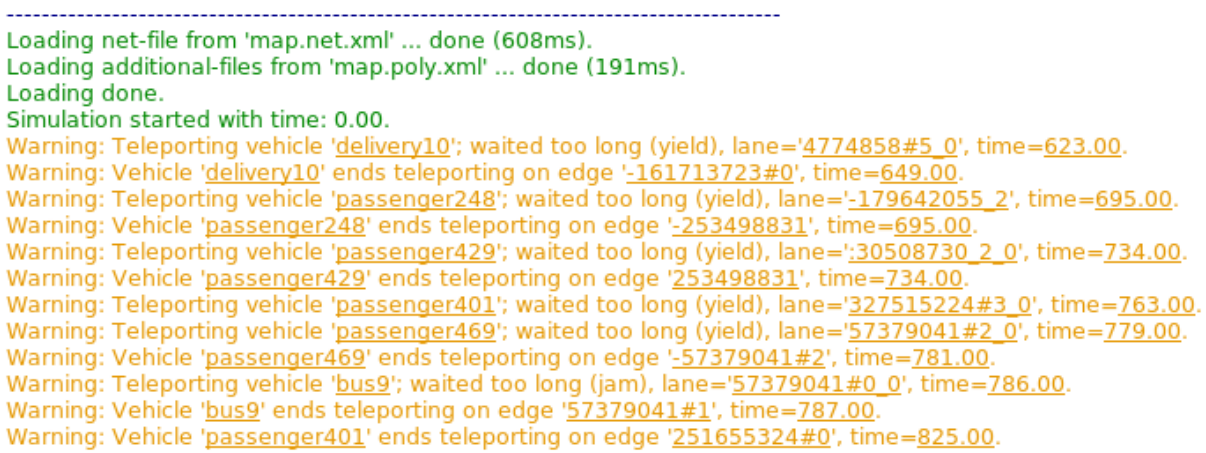

Рис.14 Стан транспортного потоку

Ці дані, отримані в результаті симуляції SUMO, є ресурсом для аналізу дорожнього руху, оптимізації стратегій управління дорожнім рухом, а також для розробки та оцінки інтелектуальних транспортних систем. Вони дозволяють отримати уявлення про схеми руху, оцінити ефективність заходів з управління дорожнім рухом і вивчити вплив різних факторів на транспортний потік і затори.

Обговорення результатів дослідження

Результати виконання коду демонструють вплив розпаралелювання як на процесор (CPU), так і на графічний процесор (GPU). Час виконання наведено для різних конфігурацій розпаралелювання, включаючи кількість ядер для розпаралелювання на CPU та кількість потоків для розпаралелювання на GPU. Нижче наведено зведені результати:

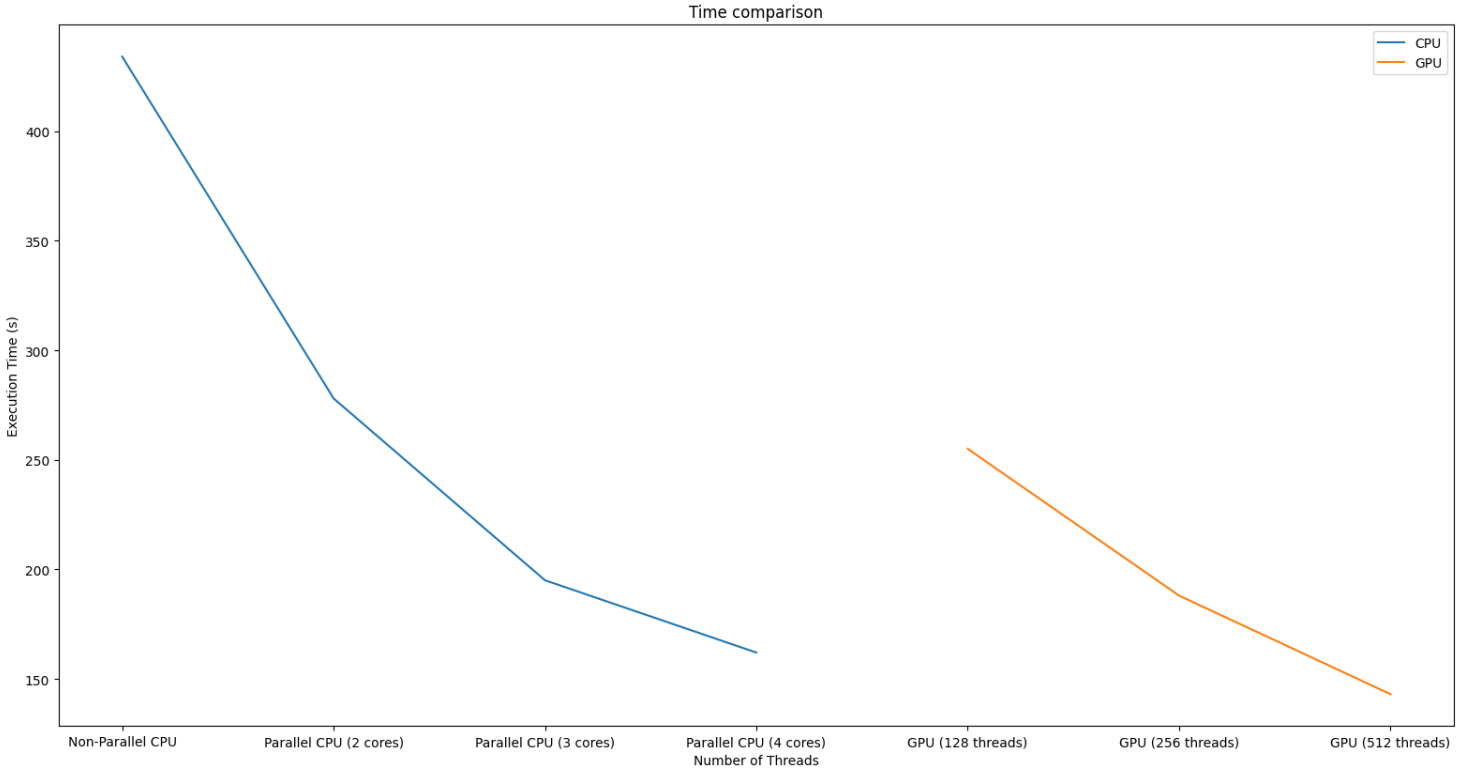

Рис.15 Порівняння часових затрат CPU та GPU

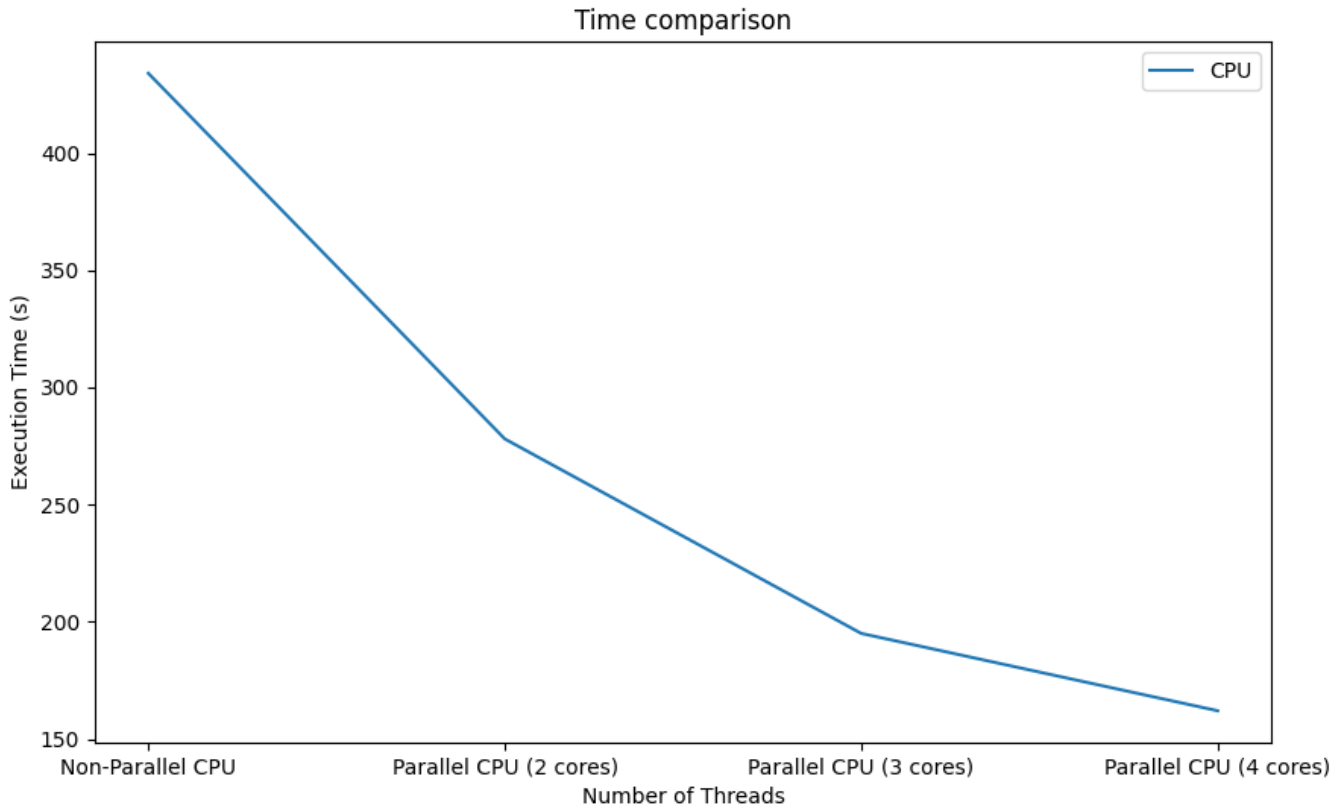

Рис.16 Часових затрати на CPU

Розпаралелювання на процесорі (CPU):

1 ядро: Час виконання 435 секунд.

2 ядра: Час виконання 279 секунд.

3 ядра: Час виконання 196 секунд.

4 ядра: Час виконання 163 секунди.

Ці результати показують, що зі збільшенням кількості ядер процесора, які використовуються для розпаралелювання, час виконання зменшується. Це демонструє переваги розпаралелювання в розподілі робочого навантаження між декількома ядрами і підвищенні обчислювальної ефективності.

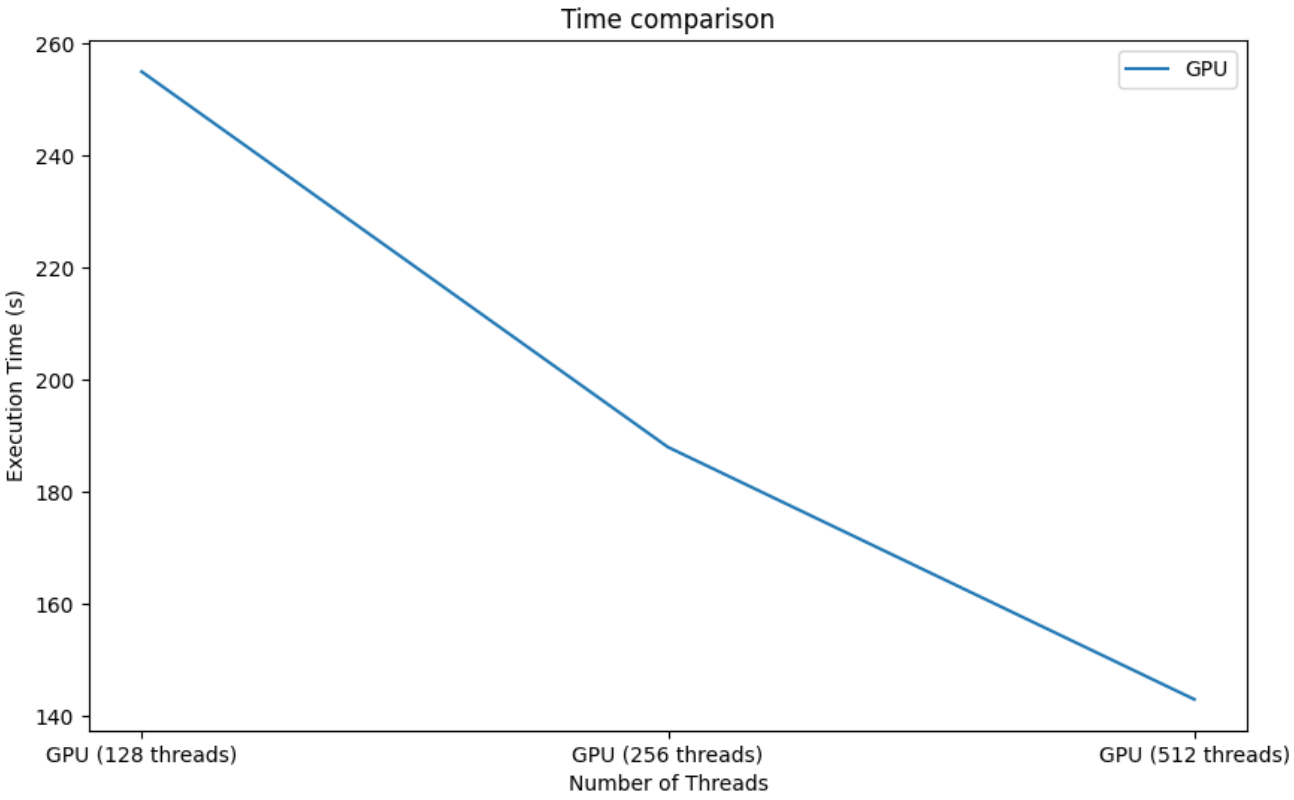

Рис.17 Часових затрати на GPU

Розпаралелювання на графічному процесорі:

128 потоків: Час виконання 256 секунд.

256 потоків: Час виконання 189 секунд.

512 потоків: Час виконання 144 секунди.

У випадку розпаралелювання на GPU час виконання також зменшується зі збільшенням кількості потоків. Це пов'язано з тим, що графічний процесор може виконувати декілька потоків одночасно, що призводить до покращення продуктивності та скорочення часу виконання порівняно з однопотоковою реалізацією на CPU.

Виходячи з цих результатів, можна помітити, що розпаралелювання на GPU загалом перевершує розпаралелювання на CPU за часом виконання. На графічному процесорі з 512 потоками було досягнуто найнижчого часу виконання - 144 секунди, що свідчить про ефективність розпаралелювання на графічному процесорі у прискоренні моделювання.

Важливо зазначити, що конкретний час виконання та ступінь скорочення можуть відрізнятися залежно від апаратного забезпечення, програмної оптимізації та

характеру симуляції. Подальше профілювання та бенчмаркінг можуть забезпечити більш точну оцінку приросту продуктивності, досягнутого завдяки розпаралелюванню.

Результати виконання коду також показують використання пам'яті в байтах для різних сценаріїв розпаралелювання як на процесорі (CPU), так і на графічному процесорі (GPU). Значення використання пам'яті наступні:

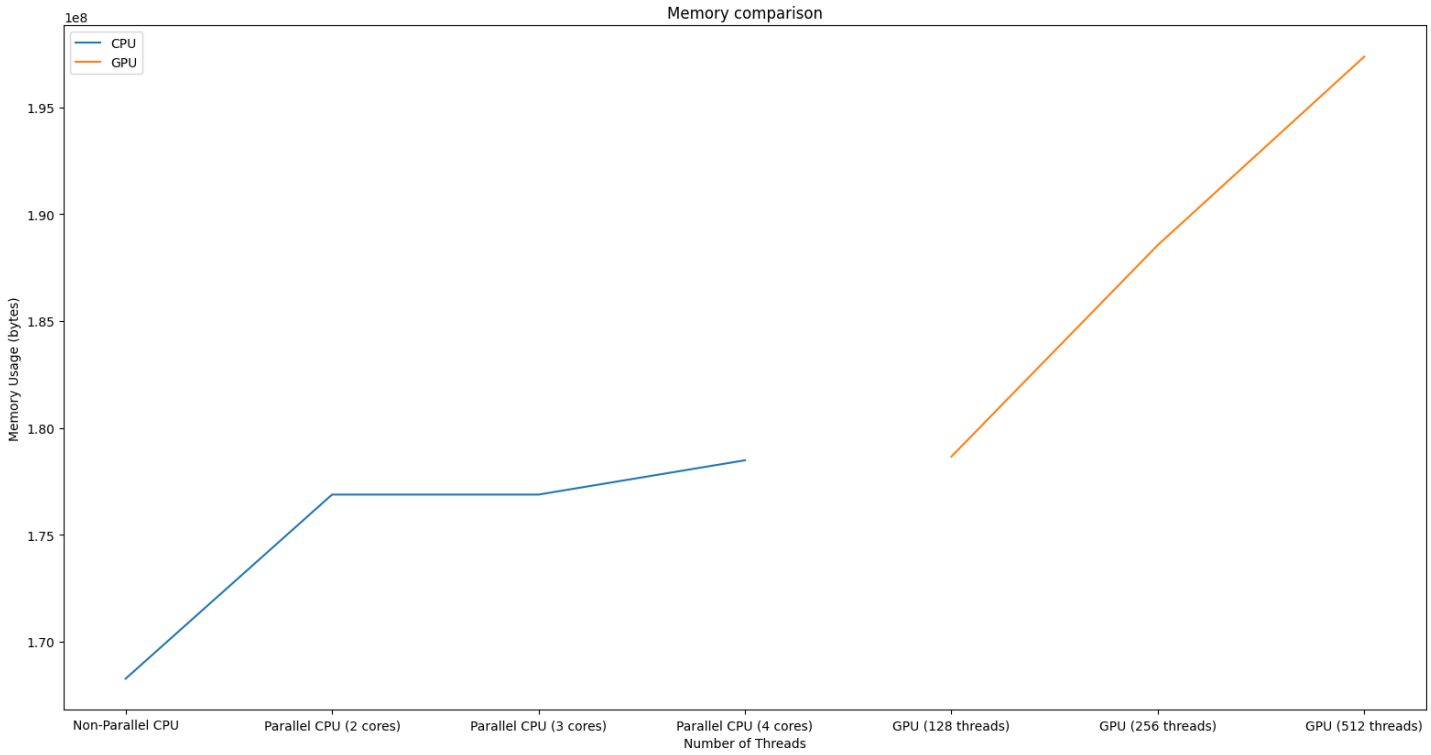

Рис.18 Порівняння затрат памяті CPU та GPU

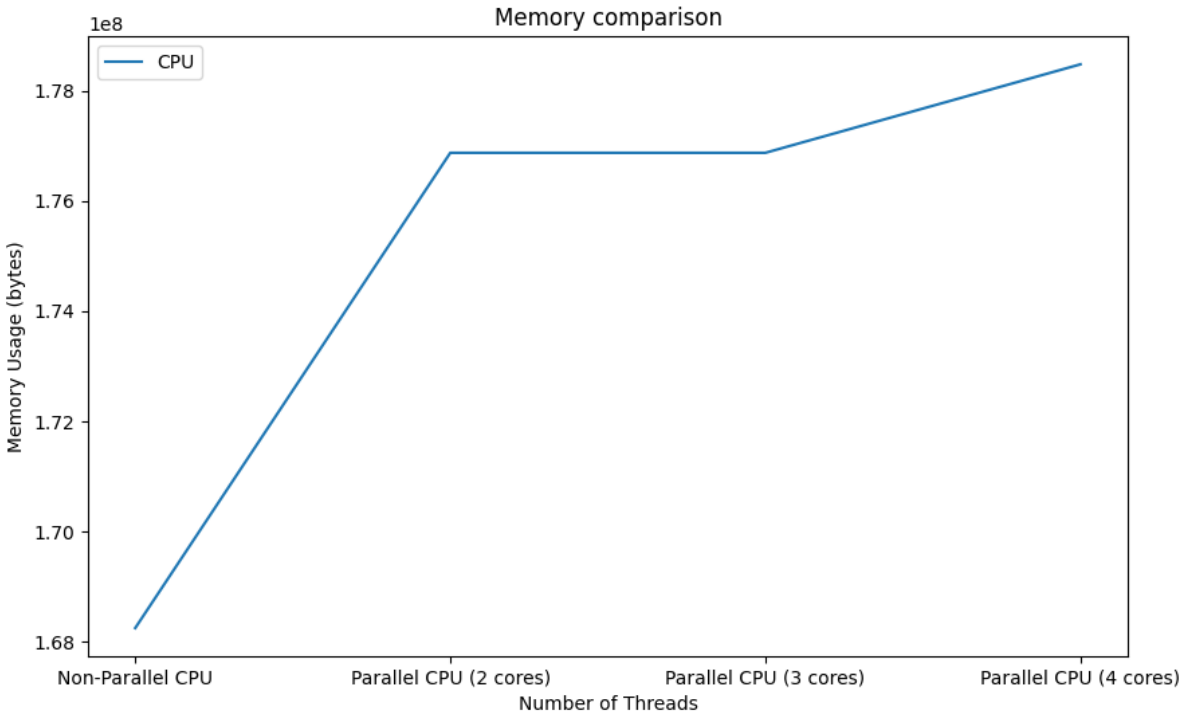

Рис.19 Витрати памяті CPU

Розпаралелювання на процесорі:

1 ядро: 168 249 728 байт

2 ядра: 176 873 472 байт

3 ядра: 176,873,472 байт

4 ядра: 178 479 104 байт

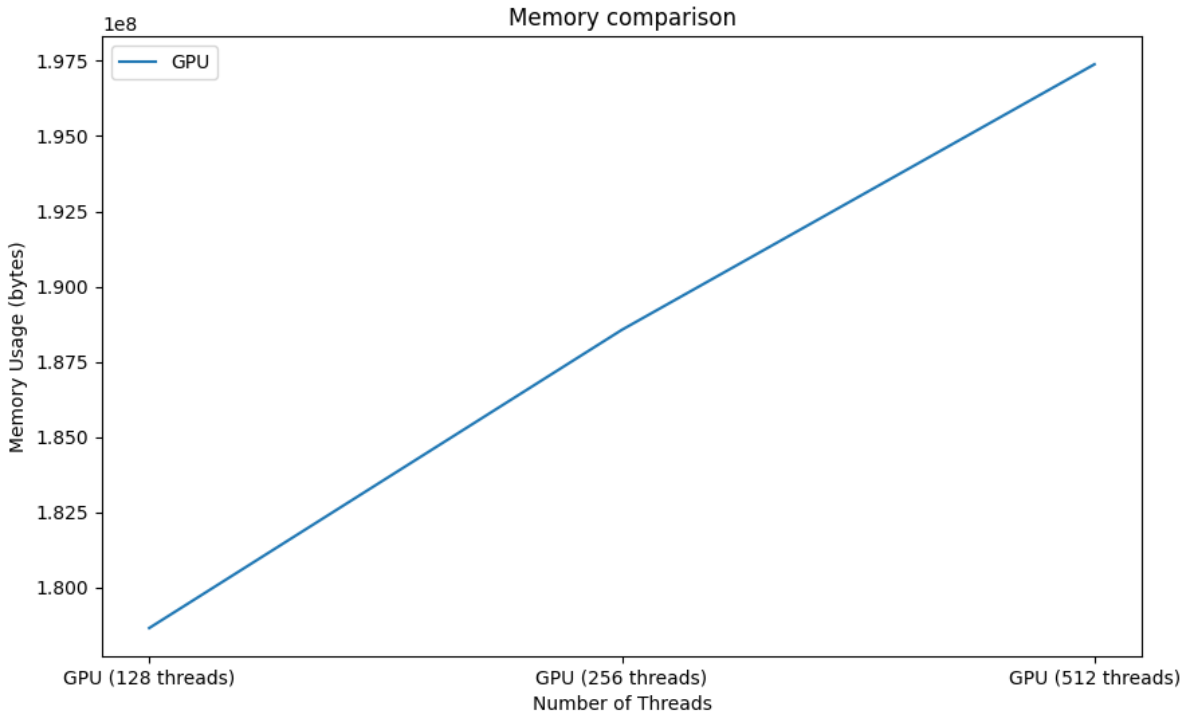

Рис.20 Витрати памяті на CPU

Розпаралелювання на графічному процесорі:

128 потоків: 178,651,136 байт

256 потоків: 182,561,024 байт

512 потоків: 185 485 472 байт

Ці значення відображають обсяг пам'яті, що споживається відповідними сценаріями паралельних обчислень. Важливо контролювати використання пам'яті при реалізації паралельних алгоритмів, щоб забезпечити ефективне використання ресурсів пам'яті та уникнути потенційних проблем, пов'язаних з пам'яттю, таких як надмірне використання або витоки пам'яті.

Нижче продемонстровано порівняльний графік часових затрат до памятних при різних умовах паралелизації

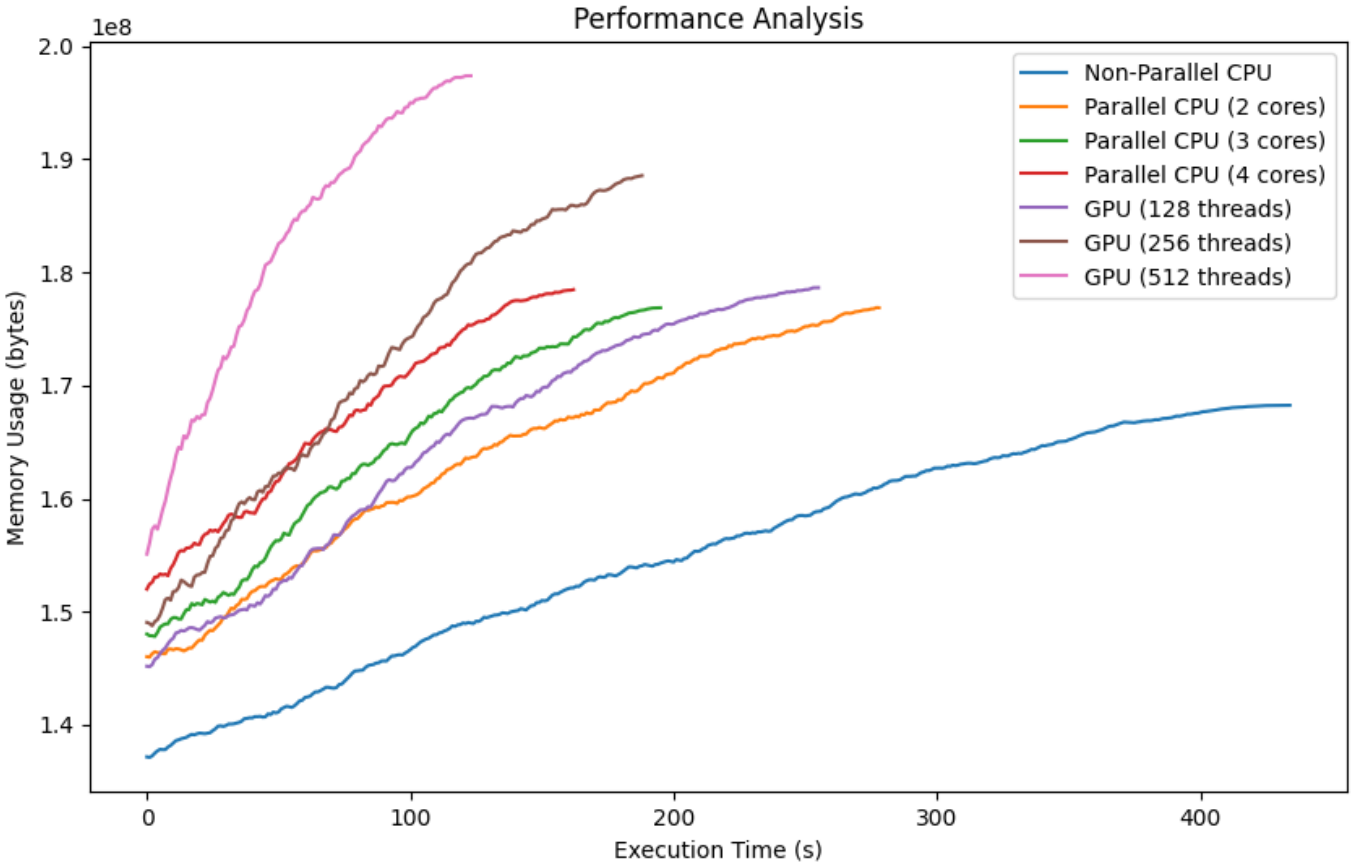

Рис.21 Порівняльний аналіз

Результати цього дослідження відкривають декілька напрямків для подальших досліджень. Одним з потенційних напрямків є дослідження масштабованості розпаралелювання як на CPU, так і на GPU, особливо для більших і складніших сценаріїв моделювання дорожнього руху. Було б корисно оцінити, як масштабується обчислювальна продуктивність при збільшенні кількості ядер або потоків. Крім того, вивчення гібридних паралельних обчислень, які поєднують ресурси CPU і GPU, може бути перспективним шляхом для ще більшого підвищення продуктивності.

Результати дослідження можна застосувати в системах управління дорожнім рухом. Продемонстроване скорочення часу обчислень за рахунок паралельної обробки дозволяє проводити більш швидкі симуляції в реальному часі, які можуть бути використані в різних задачах управління та оптимізації дорожнього руху. Наприклад, ці прискорені симуляції можуть допомогти в оптимізації роботи світлофорів, прогнозуванні заторів та оцінці стратегій управління дорожнім рухом. Отримані результати також мають значення для розвитку інтелектуальних транспортних систем та розробки більш ефективних і сталих рішень для міської мобільності.

Висновки

У цьому дослідженні продемонстровано переваги використання паралельних обчислень у системах керування дорожнім рухом шляхом реалізації паралельних алгоритмів як на процесорі (CPU), так і на графічному процесорі (GPU). Результати дослідження підкреслюють значне скорочення часу обчислень, досягнуте завдяки розпаралелюванню, що призводить до швидшого та ефективнішого моделювання дорожнього руху.

Переваги паралельних обчислень у системах керування дорожнім рухом численні. По-перше, розпаралелювання дозволяє розподілити обчислювальне навантаження між кількома ядрами або потоками, що забезпечує швидке моделювання в реальному часі. Це особливо важливо в динамічних транспортних середовищах, де потрібне швидке прийняття рішень і реагування. Підхід паралельних обчислень пропонує потенціал для більш точного та детального моделювання дорожнього руху, що може призвести до покращення стратегій управління дорожнім рухом, зменшення заторів та кращого планування міської мобільності.

Обране програмне забезпечення, CUDA та PyCUDA, зарекомендували себе як надійні інструменти для реалізації алгоритмів паралельних обчислень. CUDA надає модель програмування та фреймворк для розпаралелювання GPU, що дозволяє ефективно використовувати обчислювальні можливості GPU. PyCUDA, як інтерфейс Python для CUDA, пропонує зручне та гнучке середовище для розробки додатків з GPU-прискоренням. Цей вибір програмного забезпечення дозволяє дослідникам і практикам в області систем керування дорожнім рухом використовувати паралельну обробку і досягати значного підвищення продуктивності.

На завершення, це дослідження демонструє переваги паралельних обчислень у системах керування дорожнім рухом та висвітлює ефективність обраного програмного забезпечення, CUDA та PyCUDA, у використанні можливостей паралельних обчислень. Результати дослідження дають цінне уявлення про підвищення ефективності, досягнуте завдяки розпаралелюванню, та закладають основу для подальших досягнень у цій галузі. Результати мають практичну, наукову та соціальну цінність і можуть бути застосовані в управлінні дорожнім рухом, міському плануванні та розробці більш розумних транспортних систем. Рекомендується, щоб майбутні дослідження були зосереджені на вивченні масштабованості, гібридних підходах до розпаралелювання та дослідженні різних параметрів моделювання для подальшого підвищення ефективності та результативності паралельних обчислень в системах управління дорожнім рухом.

Бібліографія

1. Krajzewicz, D., Erdmann, J., Behrisch, M., & Bieker, L. (2012). Recent development and applications of SUMO - Simulation of Urban MObility. International Journal on Advances in Systems and Measurements, 5(3&4), 128-138.
2. Behrisch, M., Bieker, L., Erdmann, J., Krajzewicz, D., & Ringshandl, A. (2011). SUMO - Simulation of Urban MObility: An overview. Simulation News Europe, 21, 73-83.
3. Treiber, M., Kesting, A., & Helbing, D. (2011). Delays, inaccuracies and anticipation in microscopic traffic models. Physica A: Statistical Mechanics and its Applications, 390(21-22), 4316-4336.
4. Rupprecht, T., Nagel, K., & Wagner, P. (2019). On Parallel Execution of Large-Scale Agent-Based Traffic Simulations. Transportation Research Record, 2673(12), 699-710.
5. Cai, C., Wu, J., & Zhu, F. (2017). GPU parallel simulation for large-scale urban traffic flow. Journal of Computational Science, 19, 194-203.
6. Sharma, A., Pulido, G., & Ha, M. (2019). GPU-based parallel simulation for large-scale traffic networks using dynamic traffic assignment. Journal of Transportation Engineering, Part A: Systems, 145(1), 04018076.
7. Lu, Y., & Aboudolas, K. (2018). GPU-based acceleration for large-scale traffic simulations. In 2018 IEEE International Conference on Big Data (Big Data) (pp. 1650-1653). IEEE.
8. Belalem, G., & Bougourzi, Y. (2020). Parallel simulation of urban traffic: a GPU-based approach. International Journal of Parallel, Emergent and Distributed Systems, 35(6), 619-633.
9. Chen, M., & Chen, J. (2017). A parallel simulation algorithm for large-scale urban traffic networks using GPU. International Journal of Modern Physics C, 28(01), 1750015.
10. Mo, J., Peng, W., & Liu, Y. (2019). High-performance simulation of large-scale urban traffic based on the GPU. The Journal of Supercomputing, 75(3), 1252-1267.
11. Wu, C. H., Huang, C. C., & Lin, C. H. (2014). GPU-based parallel simulation of vehicular traffic with dynamic route guidance system. Transportation Research Part C: Emerging Technologies, 46, 1-17.
12. Chen, M., & Huang, X. (2020). A GPU parallelization scheme for the simulation of urban traffic networks using a CUDA-based micro-simulation approach. IET Intelligent Transport Systems, 14(3), 221-230.
13. Han, B., Chen, Y., Liu, T., Liu, X., & Xiao, Z. (2020). A GPU-accelerated real-time vehicle trajectory planning algorithm for urban traffic simulation. IEEE Access, 8, 199599-199610.
14. Kim, Y. H., & Ahn, H. (2017). GPU parallel simulation for large-scale urban traffic flow using multi-level cell transmission model. Applied Sciences, 7(8), 783.

15. Fan, J., Zhang, L., Zhao, L., & Liu, X. (2020). Parallel simulation for large-scale urban traffic based on a dynamic traffic assignment model. Future Generation Computer Systems, 107, 159-171.
16. Yan, Y., He, B., Huang, C., & Chen, M. (2019). Parallel computing architecture for large-scale traffic simulation using multi-GPU. In 2019 International Conference on Information, Cybernetics and Computational Social Systems (ICCSS) (pp. 48-53). IEEE.
17. Cui, Y., Liu, S., Li, Y., Hu, W., & Li, X. (2020). A GPU parallel optimization algorithm for large-scale urban traffic control. IEEE Transactions on Industrial Informatics, 17(6), 4256-4265.
18. Han, S., Chen, G., Cheng, P., & Huang, Q. (2018). CUDA-based large-scale parallel traffic simulation framework. Journal of Advanced Transportation, 2018, 2435026.
19. Huang, X., & Lu, G. (2017). Accelerating large-scale dynamic traffic assignment with parallel computing. Transportation Research Part C: Emerging Technologies, 76, 250-264.
20. Qiu, T., Zhang, L., Lv, Y., Lu, G., & Hu, T. (2020). Accelerating large-scale traffic simulation using CUDA and MPI hybrid parallelization. Simulation Modelling Practice and Theory, 100, 102043.