UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

Paralelización de procesos de modelamiento de tráfico urbano por medio de la contenerización del software Simulation of Urban MObility (SUMO) para Supercomputadores

TESIS PARA OPTAR AL TÍTULO DE
INGENIERE CIVIL EN COMPUTACIÓN Y
MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

PABLO VILLAR MASCARÓ

PROFESOR GUÍA:
Javier Bustos Jiménez

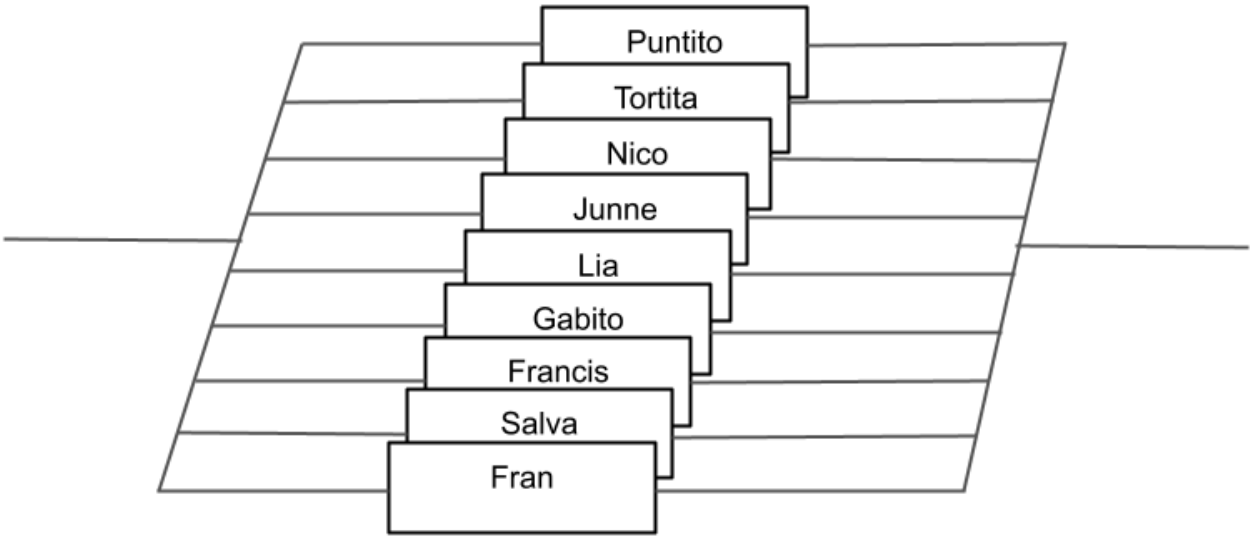
PROFESOR CO-GUÍA:
Patricio Reyes

MIEMBROS DE LA COMISIÓN:
NOMBRE COMPLETO UNO
NOMBRE COMPLETO DOS
NOMBRE COMPLETO TRES

SANTIAGO DE CHILE
2024

Resumen

Para todos los amores paralelos de mi vida



por todo su cariño y soporte a través de estos años inciertos y hostiles.

Agradecimientos

Quiero partir agradeciendo a Javier Bustos, mi profesor guía, y a Patricio Reyes, mi profesor coguía, por la oportunidad de realizar este trabajo, la paciencia, los consejos y la buena convivencia durante estos dos años de tesis. Asimismo, quiero agradecer al equipo completo de NIC Labs y del *Data Analytics and Visualization Group* del *Barcelona Supercomputing Center* por el buen recibimiento y permitirme aprender y compartir con ustedes.

Le debo la realización de este trabajo también a mi compañero de vida, Christopher Marín, Puntito, por ayudarme continuamente a depurar código, corazón y cabeza con su infinito amor, sabiduría y paciencia. Por todos estos meses de atravesar un camino incierto con la certeza de tu cariño y la esperanza de seguir construyendo juntos nuevos proyectos, nuevos juegos y nuevos horizontes.

A Fran Zautzik, por esas conversaciones interminables con un tecito en mi departamento sobre nuestros trabajos que nos trajeron tantas ideas que en cierta medida nos hicieron avanzar y por acompañarme a aguantar hasta lograr lo que había que lograr.

A mi familia, por su apoyo continuo y por soportarme en los peores tiempos para que al fin pudiera llegar a este punto.

A mi mamá, a mi papá y a mi hermana Leticia, por todo el amor por la ciencia, las preguntas, las discusiones interesantes de sobremesa; por todo este viaje que me han ayudado a emprender y por los suyos propios que de alguna manera también han sido un aporte en mi vida y nos ha llevado a tener un lugar seguro y hermoso donde poder seguir desarrollándonos.

A mi abuela Silvia, por la oportunidad de haber tenido una beca que, lamentablemente, fue otorgada a nuestra familia como «compensación» por las violaciones a nuestros Derechos Humanos. Siempre van a vivir en mí, de forma contradictoria, el agradecimiento por haber podido estudiar una carrera y la rabia que me produce la necesidad de una justicia real, con una reparación real, para que nunca más hayamos estudiantes cuya única oportunidad para estudiar en la educación superior se basa en haber tenido familiares torturados y desaparecidos. Esta es por ti, bueli, que hasta el día de hoy me enseñas lo importante que es mantenerse fiel a los principios y luchar con todo hasta el final.

A quienes se han ido dejando algo importante y valioso para ser quien soy en este momento, y a quienes a cuyo pesar tuve que seguir haciendo todo lo que logré hacer estos últimos años, gracias.

«Deja que el blanco loto florezca orgulloso,

pintado en mi espalda, mi voluntad no flaqueará.

En el cielo libre, cierro mis ojos.«

-*«Lotus»*, Dir en Grey

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

El modelamiento de tráfico vehicular urbano por medio del uso de *software* de simulación resulta ser de gran relevancia al momento de proveer información necesaria para diseños y decisiones en torno a la movilidad de distintos agentes dentro de un área determinada. En particular, se puede considerar crucial para el desarrollo de *Gemelos Digitales* de ciudades y áreas metropolitanas. Un *Gemelo Digital* consiste en «una representación de un producto activo o un sistema de productos-servicios único que incluye una selección de características, propiedades, condiciones y comportamientos por medio de modelos, información y datos a través de una o múltiples fases de su ciclo de vida» [5]. Es decir, un *Gemelo Digital* consiste en una representación en tiempo real de una entidad física en un ambiente digital [6]. En este contexto, el presente trabajo se desarrolla bajo la colaboración entre NIC Research Labs y el equipo de Data Analytics and Visualization Group del Barcelona Supercomputing Center (BSC) para la creación de una plataforma generadora de *Gemelos Digitales* para las ciudades de Kobe (Japón) y Barcelona (España), los cuales tienen por objetivo contribuir a la toma de decisiones para la implementación de políticas de urbanismo en las ciudades, con el fin de mejorar la calidad de vida mediante la ejecución de cambios estructurales en éstas.

En dicho sentido, el uso de estas herramientas de simulación cobra especial importancia al momento de querer obtener cierto nivel de precisión en el modelamiento de diferentes escenarios, tales como puntos de congestión y flujo de tráfico en horas críticas para la movilidad de las personas. Sin embargo, el obtener simulaciones detalladas a grandes escalas implica un alto costo en cuanto a recursos computacionales, por lo que durante los últimos años se ha buscado desarrollar métodos de distribución y paralelización de estos procesos, con el objetivo de optimizar los recursos disponibles para simular, por ejemplo, áreas metropolitanas grandes y concurridas.

Dada esta problemática, la presente tesis aborda la implementación de una arquitectura de paralelización y sincronización para simulaciones de tráfico vehicular urbano orientada a su uso en supercomputadores, con el objetivo de estudiar la escalabilidad de simulaciones microscópicas (es decir, de alta granularidad), las cuales son realizadas mediante el *software*

SUMO, para áreas urbanas extensas y con alta carga de tráfico vehicular. Para esto, se propone el uso de herramientas orientadas a High-Performance Computing, tales como *Singularity Containers* y *Message Passing Interface* (MPI), lo cual permite tanto la contenerización de procesos y aplicaciones complejas para su uso en supercomputadores, como la comunicación y sincronización de dichos procesos. En este caso, este trabajo se encuentra orientado a su implementación en el supercomputador *MareNostrum*, ubicado en el Centro Nacional de Supercomputación de Barcelona.

En una etapa preliminar del proyecto, se realizó un estudio sobre la escalabilidad del *software* de simulación en cuanto a los tiempos de ejecución y uso de CPU en un equipo Intel® Core™ i7-106G7 CPU @2,30GHz. Dicho estudio señaló que existe un crecimiento exponencial respecto a los tiempos de ejecución de las simulaciones ejecutadas de forma secuencial, mientras que el uso de CPU se mantenía constante en, aproximadamente, un 100%.

Posteriormente, el proyecto se dividió en tres etapas, a ser: la partición y contenerización de las simulaciones, la paralelización de las mismas, y la comunicación entre los procesos de simulación. En las tres etapas se hizo uso de *Singularity Containers*, mientras que en la tercera se sumó el uso de *OpenMP*.

Capítulo 2

Marco teórico

2.1. Simuladores de tráfico urbano

Dentro del campo de las simulaciones computacionales, se destacan las simulaciones de tráfico urbano en tanto son capaces de proveer información útil para la movilidad en distintos entornos y bajo diversas condiciones. Pablo López et. al. realizan una clasificación de los tipos de simuladores existentes según el nivel de granularidad que poseen [7]:

1. Macroscópicas, donde se modelan dinámicas a gran escala tales como la densidad total de tráfico en una ciudad.
2. Microscópicas, en las cuales se modela cada vehículo o agente como una entidad individual, con sus correspondientes interacciones.
3. Mesoscópicas, que contemplan características tanto de las simulaciones macroscópicas como microscópicas, y
4. Submicroscópicas, en las cuales no sólo se simula cada vehículo como un agente individual, sino que es posible simular dinámicas o funciones dentro de ellos.

Respecto a la variedad de simuladores de tráfico microscópicos existentes, [1] realiza una comparación entre los simuladores SUMO, Quadstone Paramic Modeler, Treiber's Microsimulation of Road Traffic, Aimsun, Trafficware SimTraffic y CORSIM TRAFVU, considerando entre sus criterios de comparación:

- Apertura de código (*open source*)
- Portabilidad de sistema operativo
- Documentación
- Escalabilidad
- *Performance*

En cuanto al criterio de apertura de código, solamente SUMO y Treiber's Microsimulation of Road Traffic resultaron ser *open source*. Esto resulta un criterio importante a considerar, dado que al poder ser modificado por otros programadores, el *software* tiene el potencial de poder paralelizar modelos de simulación y y desarrollar paquetes orientados a la alta disponibilidad. Ambos simuladores también coinciden en la portabilidad que ofrecen para

sistemas operativos Linux, mientras que el resto de *software* ofrece únicamente funcionamiento para sistemas Windows.

Respecto a la disponibilidad de documentación, la mayoría de los simuladores resultaron poseer documentación disponible al usuario, mientras que en términos de escalabilidad y performance, SUMO resulta ser el simulador con mayor capacidad de soportar grandes redes de caminos

Adicionalmente, Ejército et. al. [2] realiza una comparación entre los simuladores Matsim, SUMO, Aimsun y PTV Vissim utilizando los mismos criterios ya descritos. De estos trabajos es posible extraer la siguiente tabla comparativa entre todos los *software* presentados:

Simulador	Apertura de código	Portabilidad de SO	Documentación	Escalabilidad
SUMO	✓	Linux, Windows, MacOS	✓	Alta
Treiber's Micro-simulation	✓	Linux, Windows, MacOS	✓	Baja
Trafficware Sim-Traffic		Windows	✓	Media
CORSIM TRAF-VU		Windows	✓	Baja
Matsim	✓	Linux, Windows	✓	Media
Aimsun		Linux, Windows, MacOS		Alta
PVT Vissim		Windows	✓	Alta

Tabla 1: Comparación entre simuladores de tráfico vehicular (adaptada de [1] y [2])

En cuanto a los aspectos de *performance*, se presenta la siguiente tabla que reúne la información obtenida de [1] y [2]:

Simulador	Uso de CPU	Uso de memoria
SUMO	30-40%	12-16 MB para redes promedio
Paramics Modeler	50%	40-140 MB
Aimsun	30-40% dependiendo del número de vehículos	300-400 MB dependiendo de la red de tráfico
SimTraffic	50%	35 MB
CORSIM TRAFVU	50%	28-31 MB dependiendo de la red de tráfico
Vissim	50-60%	720 MB

Tabla 2: Comparación entre *performance* de simuladores de tráfico vehicular (adpatada de [1] y [2])

De ambas tablas presentadas, es posible concluir que SUMO parece ser una opción eficiente en cuanto a escalabilidad y performance, sumando a esto la capacidad de paralelización que implica su apertura de código y la amplia documentación que posee para el usuario.

Por otro lado, y en una comparación más detallada, Diallo et. al. [3] realiza un *benchmarking* entre los *software* Matsim, SUMO, Aimsun Next, PTV Vissim y GAMA según los criterios de naturaleza del *software*, creación de redes de caminos y demanda de transporte, realismo en las simulaciones, documentación e interfaz de usuario (GUI), y especificaciones del modelador. Respecto a este último aspecto, se desprende la siguiente tabla comparativa:

Simulador	Modelo micro/meso	Escalamiento	Salida de estadísticas	Intermodalidad	Calibración	API	Acceso a código fuente
SUMO	✓	✓	✓	✓	✓	✓	✓
Matsim	✓	✓	✓	✓	✓		✓
Aimsun Next	✓	✓	✓	✓	✓		
PTV Vissim	✓	✓	✓	✓	✓		
GAMA	✓	✓	✓				✓

Tabla 3: Comparación entre características de simuladores de tráfico vehicular (Adaptada de [3])

Dos aspectos que además destacan la diferencia entre los simuladores Matsim y SUMO radican en el modelamiento de redes multimodales y la simulación de interacciones microscópicas tales como los cruces de peatones, los cuales son atributos importantes a considerar al momento de querer recrear entornos fieles a la realidad. En este sentido, SUMO presenta ventaja sobre Matsim dado que las redes multimodales se encuentran mejor modeladas con este *software*, además de que SUMO sí presenta la capacidad de simular interacciones microscópicas en sus simulaciones.

2.2. Simulation of Urban MObility (SUMO)

Simulation of Urban MObility (SUMO) es un *software* consistente en una herramienta de simulación de tráfico continuo y microscópico diseñado para manejar grandes redes de caminos [8] y simulaciones multiagente, modelando cada vehículo de manera explícita, con rutas individuales a través de dicha red. Produce simulaciones deterministas por defecto, aunque posee opciones para introducir aleatoriedad en éstas. Este *software* provee una plataforma para simular y visualizar variados escenarios de transporte, incluyendo transporte público y privado, además de peatones y ciclistas. SUMO se encuentra diseñado para simular las redes de caminos bajo diferentes condiciones de tráfico, tales como congestión, accidentes y cambios en la infraestructura de la red. También permite el modelamiento de diferentes estrategias de gestión de tráfico e impactos de nuevas tecnologías de transporte.

Adicionalmente, este *software* se caracteriza por ser altamente portable [4], además de poseer la capacidad de importar redes de caminos de diversas fuentes, tales como OpenStreetMap (OSM), OpenDRIVE, Matsim, Vissim y otros simuladores [7]. En cuanto a la generación de rutas, SUMO permite la obtención de éstas a través de datos reales (matrices de Origen-Destino), así como la generación aleatoria de rutas a través de su herramienta *randomTrips.py*. La siguiente figura muestra el funcionamiento básico del simulador, desde los parámetros de entrada que debe obtener hasta los posibles *outputs* que dispone.

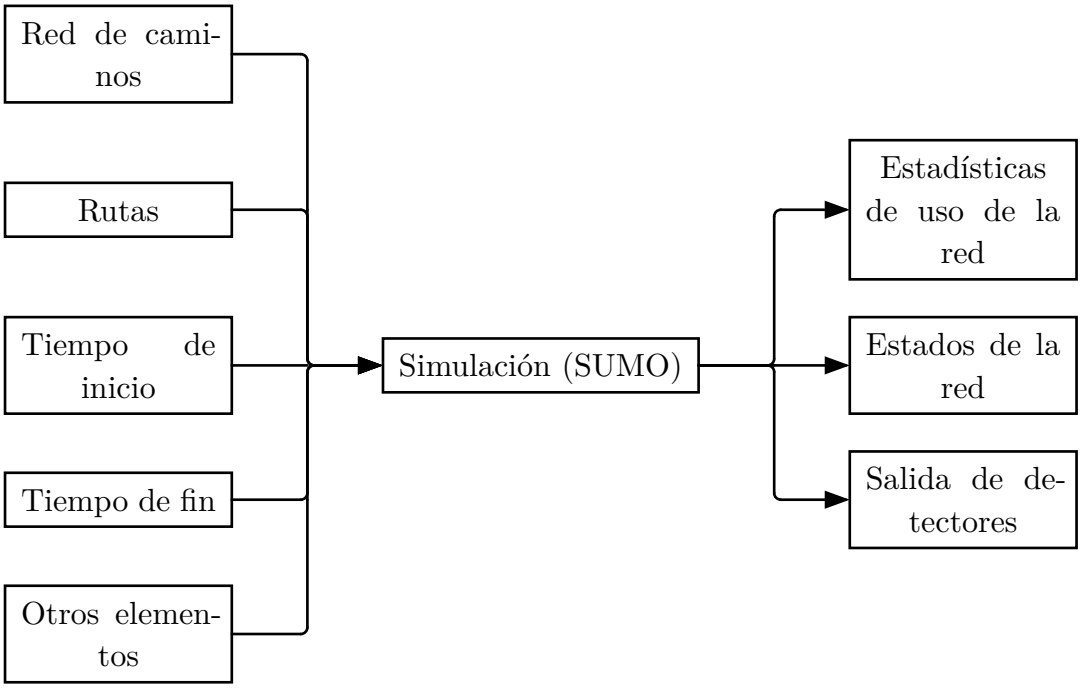


Figura 2: Entradas y salidas del simulador SUMO (adaptado de [4])

Las redes de caminos de SUMO consisten en grafos de nodos y aristas unidireccionales que representan calles y veredas. Cada arista posee una geometría descrita por un arreglo de segmentos y consisten en una o más vías corriendo en paralelo [7]. Además de estas redes, SUMO puede modelar otros elementos, tales como detectores en la red, elementos avanzados como luces de tráfico, infraestructura adicional como paradas de buses, y polígonos y puntos de interés (POI).

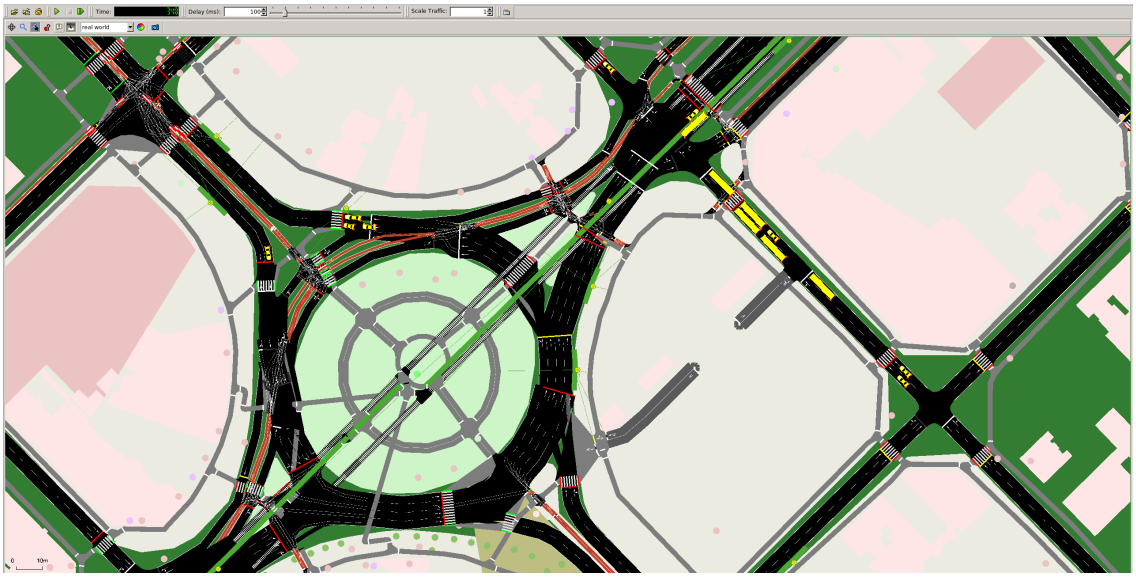


Figura 3: Ejemplo de simulación con SUMO de la ciudad de Barcelona.

Para la gestión y ejecución de las simulaciones, SUMO posee una interfaz de control de tráfico (TraCI), la cual permite obtener valores de los objetos simulados y manipular su comportamiento durante la ejecución de las simulaciones [9].

En la actualidad, SUMO también permite manejar simulaciones microscópicas de áreas extensas; diversos estudios han logrado simular ciudades tales como Luxemburgo [10] y Bologna [11]. A pesar de que estos avances en las simulaciones densas y de alta granularidad presentan sus limitaciones en cuanto a la escala de lo que se puede simular, dichos trabajos muestran que el *software* SUMO posee la capacidad de modelar entornos realistas con alta precisión.

2.3. Gemelos Digitales (*Digital Twins*) para ciudades

La revolución digital y el crecimiento global de la Internet han levantado el fenómeno de las ciudades manejadas por datos (DDC o *Data-Driven City*) y ciudades inteligentes (*Smart Cities*) [12], las cuales integran datos y tecnologías digitales para asegurar la sustentabilidad, el bienestar ciudadano y el desarrollo económico en el sector urbano.

Por otro lado, el concepto de Gemelo Digital (*Digital Twin*) conlleva el desarrollo y soporte de modelos de objetos y procesos del mundo real. En [13] se describe como «una simulación probabilística integrada multiescala de un objeto complejo que utiliza modelos físicos, matemáticos y simulativos para obtener la representación más precisa del objeto real correspondiente, basado en el análisis de datos desde diversas fuentes».

En este contexto, un Gemelo Digital de una ciudad se vale de datos de ésta, recolectados desde distintas fuentes para modelar su estructura, dinámicas y procesos. Adicionalmente, este modelo requiere la capacidad de simular escenarios hipotéticos sobre el modelo físico, ofreciendo funcionalidades que soporten cambios de estado en la infraestructura urbana y siendo capaz de proveer soluciones mediante el análisis de los datos obtenidos de los procesos simulados.

Como se define en [12], un Gemelo Digital de una ciudad es un sistema interconectado de Gemelos Digitales que representan ciertos aspectos del funcionamiento y desarrollo del entorno urbano. Por otro lado, [14] menciona que el desarrollo de los Gemelos Digitales para ciudades ha sido relativamente rápido, y se espera que provea potencial para el mejoramiento de la gestión de las ciudades y la calidad de vida de sus ciudadanos.

Parte importante de lo anterior es el poder comprender los patrones de movilidad de la gente, lo cual puede aportar a la planificación urbana y operaciones de transporte, tales como los sistemas de control de tráfico. Para esto, se hace necesario el correcto modelamiento del tráfico urbano para el desarrollo de una réplica precisa a nivel digital.

2.4. Simulaciones de tráfico con SUMO para el desarrollo de gemelos digitales

Dentro del uso de SUMO para el desarrollo de gemelos digitales, destaca el trabajo titulado *«Getting Real: the Challenge of Building and Validating a Large-Scale Digital Twin of Barcelona's Traffic with Empirical Data»* de Javier Argota Sánchez-Vaquerizo [15], el cual busca implementar una microsimulación de tráfico basada en agentes a gran escala de la ciudad de Barcelona utilizando SUMO, para mostrar las posibilidades y desafíos de construir estos escenarios basados en grandes cantidades de datos de alta granularidad, concluyendo que si bien es posible obtener simulaciones de estos escenarios, se hace necesario obtener más y mejores métodos de recolección y análisis de datos, mientras que el desarrollo de modelos más precisos y realistas sigue siendo un desafío.

Por otro lado, [16] se enfoca en la aplicación de simulaciones microscópicas en SUMO con datos reales de alta granularidad para el Gemelo Digital de la autopista de Geneva. En este trabajo, los autores destacan el mecanismo de calibración dinámica del flujo vehicular y el re-enrutamiento dinámico por medio de TraCI, características necesarias para el modelamiento de un Gemelo Digital.

Otro resultado relevante es el que presenta [17], trabajo que utiliza Gemelos Digitales para enfrentar los desafíos de performance que presenta la integración de diversas tecnologías y plataformas de datos en torno a la movilidad vehicular. Azevedo et. al. recurren al *software* SUMO para evaluar la eficiencia y confiabilidad de la arquitectura de datos vehiculares, destacando como resultado relevante la observación de que hubo un aumento significativo del uso de CPU durante la simulación, alcanzando valores máximos, como se puede observar en la siguiente tabla:

Tiempo de simulación	**Uso promedio de CPU**	**Uso máximo de CPU**	**Uso promedio de memoria**
Antes	2.3%	2,6%	1,7%
Durante	63.43%	99.9%	1.86%
Después	2.3%	2,6%	1,7%

A partir de esto, surge la necesidad de explorar vías de mejoramiento en la *performance* de las simulaciones, de manera que se optimice el uso de recursos computacionales para el funcionamiento esperado del Gemelo Digital.

2.5. Simulaciones de tráfico vehicular urbano en High-Performance Computing

Respecto al uso de simuladores de tráfico vehicular urbano en ambientes de High-Performance Computing, existe un precedente puesto por Klefstad et. al. [18] presentando ParamGrid, un framework sincronizado y escalable que distribuye una simulación a través de un *cluster* de *performance* ordinaria, compuesto por computadores personales de bajo costo. Este trabajo utiliza el simulador PARAMICS y se muestra que el *speedup* de las simulaciones aumenta de forma lineal de acuerdo al número de computadores presentes en el *cluster*. Sin embargo, y dadas las limitaciones que presenta, se señala como trabajo futuro el mejoramiento del algoritmo de partición de las redes de caminos basado en la carga computacional anticipada, además de establecer como necesario el estudio sobre la distribución dinámica de carga vehicular y la implementación de enrutamientos globales más eficientes.

Más adelante, Suzumura et. al. presentan una plataforma de simulaciones de tráfico multi-modal y altamente escalable [19]. Si bien el estudio de *performance* correspondiente demuestra que el simulador es 15.5 veces más rápido que el tiempo real con 12 *threads* en paralelo, se propone como trabajo futuro una mejora dentro de la precisión de las simulaciones por medio de la calibración de los agentes y la introducción de la búsqueda del camino más corto en relación al tiempo de viaje para los algoritmos de enrutamiento. Fuera de ello, no se hace mención a la portabilidad del simulador para otros entornos de High-Performance Computing.

Por otro lado, Umemoto et. al. realizan una serie de simulaciones implementadas en SUMO sobre la ciudad de Kobe para su ejecución en el supercomputador K [20]. Los resultados arrojan que una demanda alta de simulaciones requiere un mayor tiempo para empezar a ejecutarse, mientras que los tiempos de simulación llegaron a ser incluso el doble de lo que toma la misma simulación en una CPU convencional.

Finalmente, durante el año 2021 se presenta el trabajo de Franchi et. al., el cual implementa un pipeline para simulación de vehículos autónomos utilizando Webots y SUMO en paralelo en un *cluster* HPC (Palmetto Cluster) [21]. Si bien los resultados muestran una mejora en la performance de las simulaciones multimodales, se destaca que es necesario tener en cuenta el *overhead* que implica el levantamiento de las simulaciones y la escritura de resultados por cada simulación ejecutada en paralelo.

2.6. Paralelización de SUMO

Durante los últimos años, han surgido distintas técnicas y herramientas que podrían facilitar las simulaciones de tráfico en paralelo aplicado a ambientes de High-Performance Computing, tales como supercomputadores. Algunas de estas son:

1. **Paralelización híbrida**: consiste en la combinación de múltiples técnicas de paralelización, incluyendo estándares de transmisión de mensajes para computación paralela tal como lo es MPI (*Message Passing Interface*), con el objetivo de alcanzar una mejor performance y escalabilidad. Esto permite una distribución más granular de tareas de simulación a través de múltiples nodos.
2. **Particionamiento de modelos**: refiere a la partición de modelos de simulación en submodelos que pueden ser ejecutados en paralelo, lo cual puede ayudar en reducir los costos de comunicación y mejorar la escalabilidad de las simulaciones.
3. **Simulaciones cloud-based**: implica la ejecución de modelos de simulación en Cloud Computing, lo cual ofrece la oportunidad de tener simulaciones más flexibles y escalables. Sin embargo, la latencia de la red y las posibles amenazas en torno a la seguridad de los datos son desventajas que esta opción presenta.

Un estudio que aborda el problema de la paralelización de SUMO refiere a [22], en donde se presenta un modelo de particiones de mapa simplificadas para la simulación en paralelo de tráfico en zonas urbanas. Este modelo consiste en la división de las redes de caminos en forma de cuadrículas, lo cual incrementa el nivel de error de la simulación de manera proporcional con el número de particiones realizadas. Además de esto, dicho estudio señala que el método de paralelización presentado es más lento que su contraparte centralizada dado el proceso de sincronización implementado.

Por otro lado, el trabajo titulado «QarSUMO: A Parallel, Congestion-optimized Traffic Simulator» [23], presenta una versión paralelizada de SUMO considerando la complejidad de la distribución de las redes de caminos por medio de la implementación de un esquema de partición de grafos no-regulares, abreviado como METIS [24]. Si bien este estudio logra establecer un método de paralelización de alto nivel, incrementando la eficiencia de las simulaciones para situaciones de alta congestión de tráfico, su alcance en cuanto a áreas urbanas no alcanza una escalabilidad a nivel de simulaciones completas al no considerar su implementación en ambientes de *High-Performance Computing* tales como supercomputadores. Sumado a esto, el *software* disminuye el nivel de granularidad de las simulaciones por medio de la simplificación del modelamiento de un gran número de vehículos a una sola entidad.

2.7. Pruebas de escalabilidad en SUMO

En un estudio preliminar al desarrollo de este trabajo de tesis, se realizaron distintas pruebas de escalabilidad para observar el crecimiento de los tiempos de ejecución de SUMO en un computador personal. En particular, se realizaron dos pruebas de escalabilidad considerando dos aspectos importantes de los requerimientos para la creación de simulaciones de tráfico urbano: en primer lugar, se considera el tamaño de la red de caminos en número de aristas del grafo que representa el mapa, mientras que para el segundo se toma en consideración el número de agentes o vehículos por simulación.

Ambos experimentos se encuentran implementados en el lenguaje de programación C++ y fueron testeados en un computador personal con Intel(R) Core(TM) i7-106G7 CPU

@1,30GHz. Para cada test se asignó únicamente una CPU, con el objetivo de mantener la simplicidad de los resultados. Adicionalmente, sólo fue considerado un tipo de agente para las simulaciones (automóviles) y cada simulación representa 1 hora de tráfico continuo en la ciudad de Barcelona, España, ciudad cuya área es de aproximadamente 101,9 km².

2.7.1. Prueba de escalabilidad basada en el tamaño de la red para computadores personales

Para este experimento, se implementa una forma automatizada para obtener los datos necesarios desde OpenStreetMap (OSM). Esta automatización toma una localización descrita en coordenadas, y consulta los datos requeridos a través de 30 iteraciones, donde el temaño de la red es más grande con cada iteración.

Luego, se crean las simulaciones correspondientes mediante el *software* SUMO. Los archivos que describen los viajes, es decir, los puntos de partida y llegada de cada vehículo, fueron creados con *randomTrips.py* por simplicidad [7], haciendo uso de una semilla de aleatoriedad definida; en tanto, los archivos correspondientes a la descripción de las rutas fueron creados mediante el uso de la herramienta *duarouter* de SUMO [25]. Finalmente, se midió el tiempo de ejecución de cada simulación versus el número de aristas del grafo correspondiente. El pipeline para la generación de estas simulaciones se muestra en la siguiente figura:

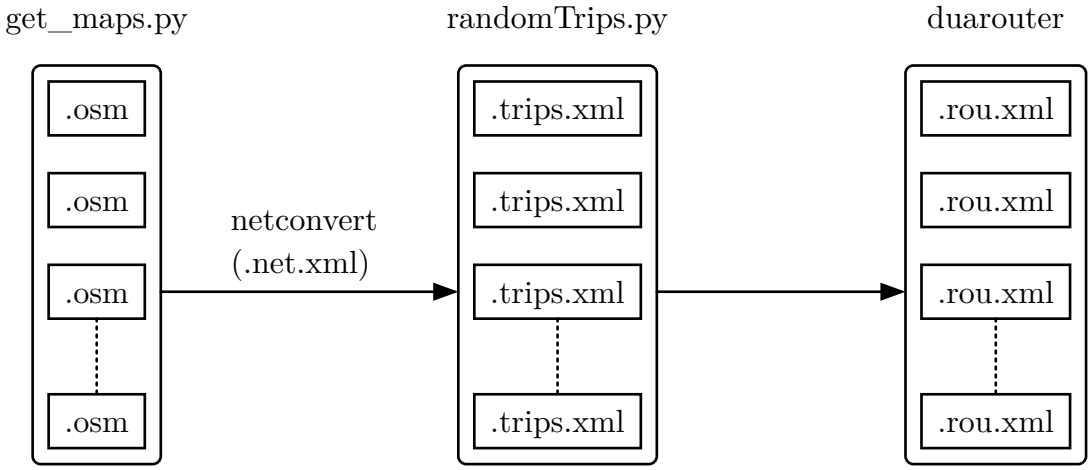


Figura 4: Pipeline para simulaciones basadas en el tamaño de la red.

2.7.2. Prueba de escalabilidad basada en el número de agentes por simulación para computadores personales

Por otro lado, en orden de testear la escalabilidad de las simulaciones respecto al número de agentes, se determina una red de caminos de tamaño fijo sobre la ciudad de Barcelona y se crean las simulaciones correspondientes con *randomTrips.py* y *duarouter*. El pipeline para este experimento se muestra en la Figura 5.

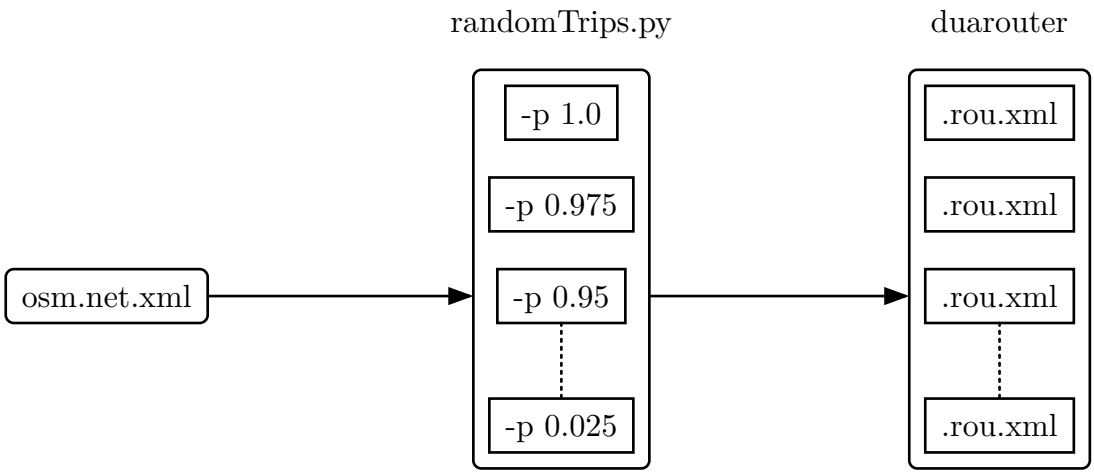


Figura 5: Pipeline para simulaciones basadas en la carga de tráfico vehicular.

Para poder incrementar de manera uniforme la cantidad de vehículos presentes en la simulación, se configuró la tasa de inserciones de agentes de *randomTrips.py* [26] para tomar valores decrecientes en el intervalo [0.25, 1.0], mientras que la configuración de *duarouter* no fue alterada.

En total, 40 simulaciones con diferentes tasas de inserción fueron realizadas, midiendo el tiempo de ejecución de cada una. El tiempo total destinado a ejecutar todas estas simulaciones fue de aproximadamente 48 horas cronológicas.

2.7.3. Resultados

Los resultados obtenidos para ambas pruebas se muestran en los gráficos expuestos a continuación. A la izquierda, se muestran los tiempos de simulación en función del número de aristas de la red, mientras que a la derecha, se muestra el uso de CPU por cada simulación ejecutada.:

Resultados para simulaciones basadas en el tamaño de la red:

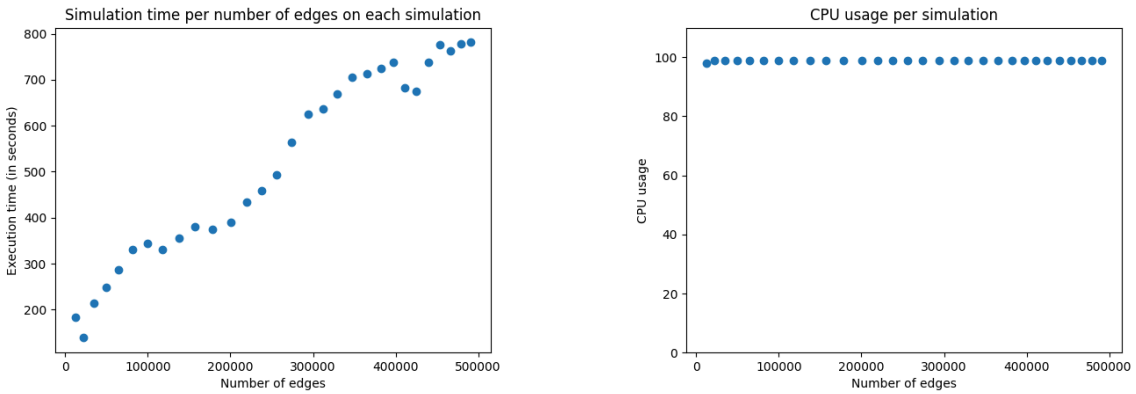


Figura 6: Resultados para la prueba de escalabilidad basada en el tamaño de la red de caminos.

Como es posible observar, en este caso existe un crecimiento lineal de los tiempos de simulación en relación al crecimiento de la red de caminos, lo cual nos muestra que el dicho

crecimiento no es realmente un problema al momento de ejecutar simulaciones sobre áreas extensas. Aún así, cabe destacar que el uso de CPU se mantiene constante en un valor máximo.

Resultados para simulaciones basadas en la carga de tráfico vehicular:

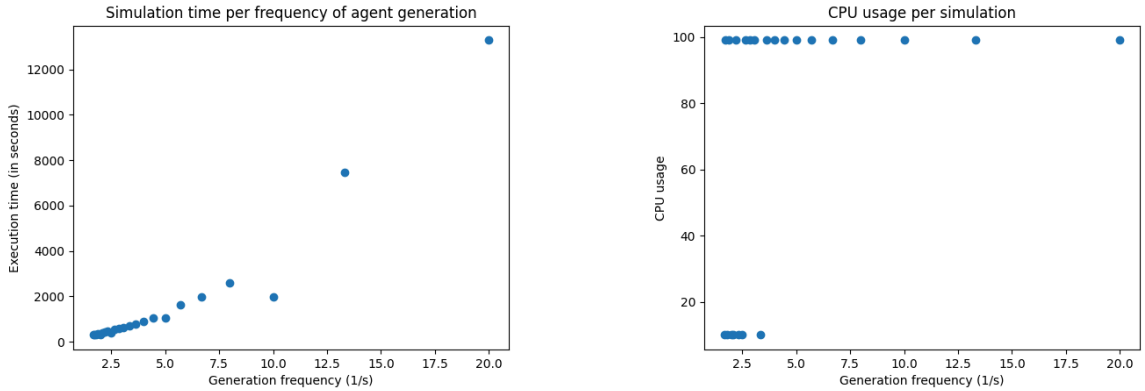


Figura 7: Resultados para la prueba de escalabilidad basada en la carga de tráfico vehicular.

En este caso, se puede observar un crecimiento exponencial de los tiempos de ejecución de cada simulación en relación a la frecuencia de inserción de vehículos en ésta. Es relevante mencionar que, a pesar de aumentar los tiempos de manera exponencial, las simulaciones no realizaron uso de la memoria *swap* del computador, lo que deja en evidencia que el problema principal para la ejecución de las simulaciones en términos de procesamiento refiere a la carga de vehículos o agentes dentro de la simulación más que el tamaño de la red, y sugiere la necesidad de paralelización de estos procesos. Por otro lado, se obtiene que el uso de CPU también se mantiene constante en un valor máximo para la mayoría de los casos.

Capítulo 3

Problema

Como es posible observar a partir de lo ya expuesto, existen principalmente dos dificultades al momento de buscar mejorar la *performance* de las simulaciones de tráfico urbano. La primera de ellas, y la más importante, refiere a la cantidad y variedad de agentes presentes en cada simulación. Dado que de la ejecución secuencial se tiene que los tiempos de ejecución aumentan de forma exponencial, al momento de querer simular redes de caminos con alta congestión de tránsito, el tiempo de ejecución de la simulación se alejará cada vez más de lo que se desea como una simulación en tiempo real.

Otro problema a considerar tiene que ver con la paralelización misma y la sincronización de los procesos paralelos, dado que existe la posibilidad de generar cuellos de botella en las comunicaciones entre nodos y de esta forma perjudicar la *performance* de las simulaciones.

El problema que se plantea a partir de esto radica en la necesidad de paralelizar simulaciones que no sólo representen grandes áreas metropolitanas, sino que también presenten una alta carga de tráfico de agentes (ya sea vehicular o de peatones), manteniendo la granularidad y precisión de dichas simulaciones en un ambiente de supercomputación y mejorando el *speedup* de éstas de manera significativa, reduciendo también los posibles cuellos de botella en la comunicación de los procesos paralelizados.

Capítulo 4

Preguntas de investigación, hipótesis y objetivos

4.1. Preguntas de investigación

- ¿Cómo se diferencian en cuanto a *performance* la versión secuencial de simulaciones en SUMO respecto a una versión paralelizada en un ambiente de supercomputación?
- ¿Es posible mejorar la *performance* de simulaciones de tráfico urbano en orden de poder simular áreas metropolitanas completas a nivel microscópico?
- ¿Cómo se comporta asintóticamente el área a simular en un ambiente de supercomputación?
- ¿Cómo se comporta asintóticamente la carga vehicular a simular en un ambiente de supercomputación?

4.2. Hipótesis

Es posible aumentar la escalabilidad de simulaciones de alta granularidad mediante la paralelización y sincronización de los procesos del *software* SUMO en un ambiente de supercomputación, incrementando el *speedup* de éstos en al menos un 5% de los tiempos de simulación secuenciales, y optimizando así el uso de recursos para simulaciones que contemplen áreas metropolitanas de gran extensión y alta carga de tráfico vehicular.

4.3. Objetivos

4.3.1. Objetivo General

Avanzar en el estado del arte acerca de la paralelización de procesos de simulación de tráfico urbano en supercomputadores, comprobando el aumento en la escalabilidad de dichos procesos luego de su correspondiente paralelización.

4.3.2. Objetivos específicos

1. Aplicar la contenerización de SUMO en *Singularity Containers* para su ejecución en supercomputadores.

2. Diseñar e implementar un modelo de paralelización de los procesos contenerizados, determinando los parámetros necesarios para la efectiva comunicación y sincronización entre los procesos.
3. Medir la escalabilidad de la solución implementada en un ambiente de supercomputación y comparar con las mediciones previamente realizadas.

Capítulo 5

Metodología

5.1. Herramientas utilizadas

5.1.1. *Singularity Containers*

Singularity es un proyecto *open-source* que permite la creación de contenedores de forma portable y reproducible [27]. A diferencia de *Docker*, *Singularity* fue creado para ejecutar aplicaciones complejas en *clusters* HPC, cobrando popularidad de manera significativa para su uso en supercomputadores; sin embargo, es posbile construir contenedores de *Singularity* a partir de contenedores de *Docker*.

Adicionalmente, este *software* soporta la generación de imágenes *custom* a través de sus archivos de definición, permitiendo envolver todas las imágenes necesarias en un sólo archivo que provee facilidad de manejo, además de simplificar el despliegue de los contenedores [28]. La interoperabilidad que ofrece *Singularity* permite también su portabilidad a través de distintas arquitecturas HPC.

5.1.2. *OpenMP*

OpenMP es una API de modelo escalable que ofrece una interfaz flexifle y sencilla de manejar para programación paralela [29]. Provee soporte para los lenguajes `C`, `C++` y `Fortran`, y permite el desarrollo de aplicaciones portables, encontrándose orientado principalmente a la programación para multiprocesadores de memoria compartida.

Esta API consiste en una especificación para un set de directivas de compilación, rutinas de librería y variables de ambiente que pueden ser usadas para especificar paralelismo de alto nivel. Actualmente, se encuentra en la versión 5.2.

5.1.3. SUMO y *libtraci*

La librería *libtraci* de SUMO es una herramienta que compatibiliza el uso de TraCI con códigos de C++ [30], de manera que no sólo se pueda ejecutar una simulación, sino que también pueda controlarse y monitorearse sin necesidad de una GUI. *Libtraci* provee, además de las funcionalidades de TraCI, soporte para el uso de múltiples clientes, lo cual resulta conveniente al momento de pensar en la paralelización de las simulaciones.

5.2. Partición de mapas de Barcelona y Viladecans

Para realizar la división de los mapas originales de Barcelona y Viladecans se utilizó una implementación en `Java` ya existente del algoritmo de *SPartSim* [31], el cual ejecuta una división por zonas geográficas eligiendo puntos distantes en el mapa dada una heurística inicial y va expandiendo las particiones con los nodos cercanos en cada iteración, para luego balancear la carga de las particiones moviendo aquellos nodos con más peso a particiones vecinas que tengan un menor peso. El objetivo de esto es entregar un mapa particionado de manera relativamente equitativa, de forma que los hilos de ejecución de cada sub-simulación no se vean sobrecargados.

En orden de mantener la compatibilidad con los archivos de entrada de las simulaciones con SUMO, a dicha implementación se le agregó un módulo de comptabilidad con archivos XML, ya que el código original sólo trabajaba con archivos `GeoJSON`, los cuales además perdían información importante al momento de convertir desde archivos `XML`. A raíz de esto, se debió implementar una clase que conservase dicha información, para después restaurarla al momento de hacer la conversión desde los grafos generados por el *software* hacia la clase de grafos compatibles con la estructura `XML`requerida por las simulaciones. La siguiente figura ilustra el proceso de conversión implementado para la compatibilidad del *software* de particionamiento de rutas con archivos `XML`.

5.3. Generación y partición de rutas vehiculares

Para la generación de las rutas vehiculares se consideraron, en primer lugar, dos enfoques distintos: por un lado, el generar dichas rutas a partir de datos reales de movilidad urbana, y por otro lado, generarlas de manera aleatoria o pseudo-aleatoria por medio de la herramienta *randomTrips.py*, provista por SUMO y utilizada para realizar las pruebas de escalabilidad de las simulaciones secuenciales.

Respecto al primer enfoque, se obtuvieron los datos de movilidad de la población de Barcelona en formato de matrices de Origen-Destino obtenidas a partir de los datos recolectados de las redes móviles por parte del ayuntamiento de Barcelona. Sin embargo, por simplicidad para la generación de las rutas, se decidió tomar el segundo enfoque.

Para la partición de las rutas generadas se hizo uso de la herramienta *cutRoutes.py* de SUMO, la cual corta las rutas acorde a la partición del mapa que se busca simular. No obstante, dicha herramienta presenta problemas en tanto termina descartando ciertas rutas particionadas y recalculando los tiempos de partida para cada una; si bien existe una extensión que soluciona este problema para la futura paralelización de los procesos de simulación [32], no fue posible acceder a ella, por lo que sólo se consideraron aquellas rutas resultantes de *cutRoutes.py* para la posterior reconstrucción.

5.4. Contenerización de simulaciones secuenciales

En una primera instancia de exploración del uso de Singularity Containers, se procedió a implementar la contenerización de simulaciones sin paralelizar. El resultado de esto fue la creación de un archivo de definición de *containers* con extensión `.def`, el cual construye el

entorno necesario para la ejecución de las simulaciones de manera aislada. A continuación, se muestra un ejemplo de estos archivos de definición para el levantamiento de los contenedores.

```
Bootstrap: debootstrap
OSVersion: jammy
From: ubuntu:22.04
MirrorURL: http://us.archive.ubuntu.com/ubuntu/

%files
    partition.net.xml /home/
    viladecans.poly.xml /home/
    partition.rou.xml /home/
    sim.sumocfg /home/
    traci_simulation /home/

%post
    chmod 755 /root
    apt update
    apt-get install -y software-properties-common build-essential python3
    add-apt-repository universe
    add-apt-repository multiverse
    apt-get update
    apt-get install -y sumo sumo-tools sumo-doc


%environment
    export SUMO_HOME=/usr/share/sumo

%runscript
    cd /home
    ./traci_simulation"
```

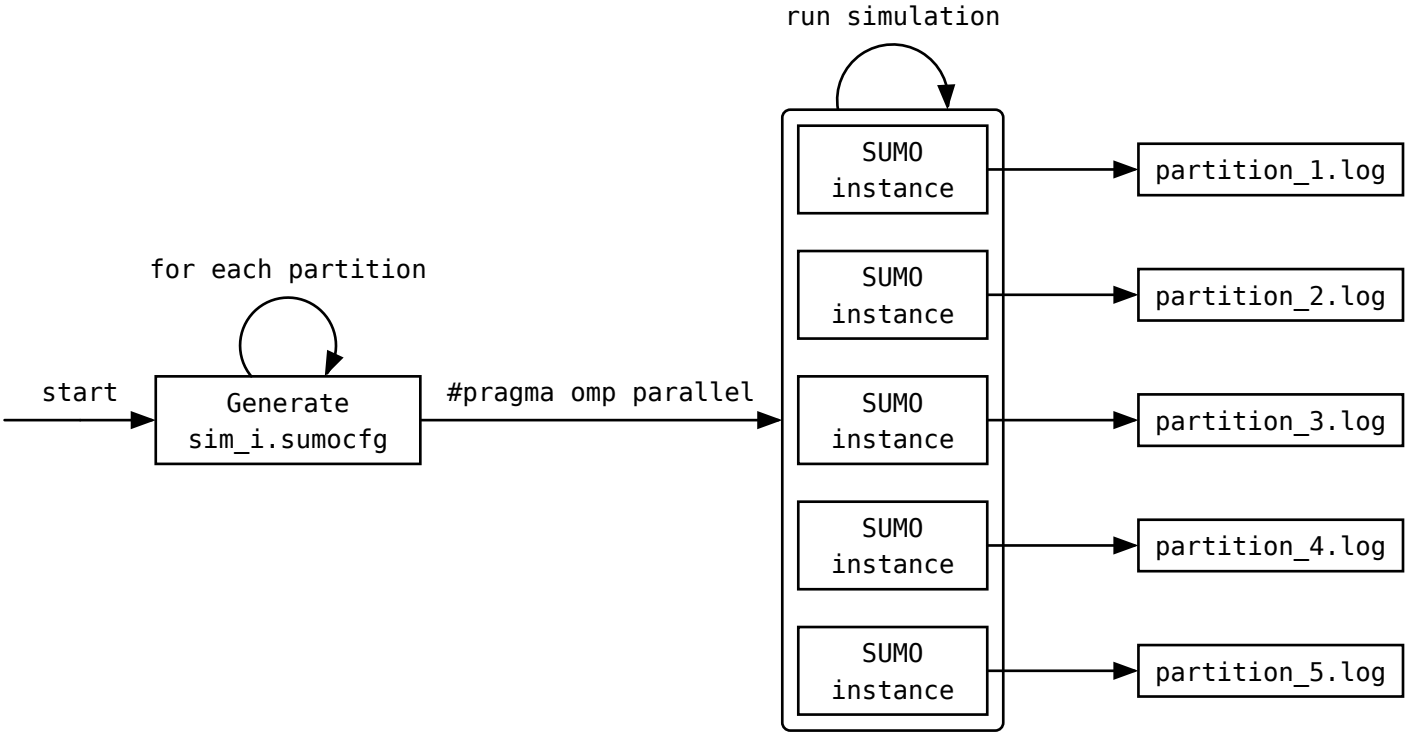
El *script* de C++ *traci_simulation* tiene por objetivo ejecutar simulaciones aisladas de forma secuencial. Para esto, luego de configurar el entorno de la simulación, el código llama a una instancia de SUMO con el archivo de configuración previamente definido a partir de un *script* en *Python*, el cual genera un archivo `XML` especificando, entre otras cosas, cuáles son los archivos a utilizar para las redes de caminos y las rutas de los vehículos.

5.5. Paralelización de las simulaciones particionadas

La paralelización de las simulaciones particionadas consistió en la implementación de un código en `C++` con la ayuda de *OpenMP* para las directivas de compilación. Este código implementa una arquitectura de *multi-threading*, en la cual cada hilo de ejecución posee una instancia de SUMO contenerizada que simula una partición dada del mapa original, con las rutas propiamente particionadas de acuerdo con el algoritmo de *cutRoutes.py*.

En la siguiente figura, se muestra el funcionamiento general de la paralelización realizada con 5 threads. Primero se realiza un proceso de configuración de forma secuencial, en el cual se generan los archivos de configuración *sim.sumocfg* a partir de los cuales se ejecutan

las simulaciones en forma paralela mediante una instancia de SUMO contenerizada con *Singularity*. De esta manera, una vez que se crean los archivos de configuración, se realiza un llamado con el comando `#pragma omp parallel`, configurando el número de threads a ejecutarse como el número de particiones hechas previamente con *SPartSim*.



5.6. Sincronización de las simulaciones particionadas

5.6.1. Reensamblaje de rutas

En cuanto a la reconstrucción de las rutas, se implementó un código en *Python* que toma tanto las rutas originales como las particionadas y, por cada ruta particionada, busca la ruta original y las agrupa en un diccionario indicando sus aristas, la partición a la que pertenece la ruta, y su índice de ocurrencia en la ruta original. De esta manera, fue posible realizar una reconstrucción parcial de los viajes generados por *randomTrips.py*.

El resultado de esta reconstrucción se guarda finalmente en un archivo `.json`, con la estructura que se muestra a continuación:

```
{
    "routes": [
        {
            "original_route": str,
            "cut_routes": [
                {
                    "partition": int,
                    "id": str,
                    "cut_route": str,
                    "next_partition": int,
                    "next_route": str
```

```
                },
                ...
            ]
        },
        ...
    ]
}
```

5.6.2. Comunicación entre nodos

Dado que la implementación sigue un enfoque para *shared-memory devices*, la comunicación entre los nodos de ejecución se realizó mediante la implementación de una cola global en la cual, a cada paso de la simulación, cada hilo escribe los vehículos salientes de su partición con la partición a la que corresponde insertarse después, mientras que al paso siguiente lee e inserta en la simulación aquellos vehículos cuya entrada corresponde a su partición. Para esto, se hace uso de exclusión mutua por medio de la implementación de secciones críticas para la escritura en la cola de tráfico, mientras que para la adición o remoción de vehículos en cada partición se hace uso de las herramientas provistas por *libtraci*.

5.6.3. Impedimentos para la sincronización

Como se menciona en la sección 5.3, la herramienta de particionamiento de rutas *cutRoutes.py* presenta limitaciones al momento de considerarla para la sincronización de simulaciones de tráfico urbano mediante SUMO, principalmente al momento de definir los tiempos de partida de los vehículos (*departure times*), redefiniendo aquellas rutas que cruzan más de una partición como rutas independientes con el mismo tiempo de partida; esto radica en problemas tales como que al ingreso de nuevas rutas, no se reconozca el vehículo a insertar dado que el mismo ya ha salido de la simulación.

5.7. Diseño e implementación de test de carga

5.7.1. Test versión secuencial

Para poder realizar la comparación en cuanto al crecimiento de los tiempos de ejecución para las simulaciones secuenciales, se implementó un módulo de test en el lenguaje `C++`, el cual contempla una primera fase de *setup*, donde se definen los períodos de generación de vehículos a partir de los cuales se generan los archivos de rutas necesarios para el mapa definido. En total, se definen veinte períodos para los cuales se definen también los archivos de configuración a utilizar para cada simulación.

La segunda etapa contempla de ejecución de cada simulación de forma secuencial, midiendo y registrando el tiempo que toma cada una en finalizar. Para esto, cada simulación se ejecutó un total de cincuenta veces, para luego obtener el tiempo total como el promedio de los tiempos de cada iteración.

5.7.2. Test versión paralelizada

Para probar el escalamiento de la solución implementada, se siguió un procedimiento similar a los test secuenciales, resultando en la implementación de los siguientes módulos:

- ***setup***: este módulo genera los períodos, rutas, particiones y archivos de configuración necesarios para cada simulación, y organiza los archivos en sus directorios correspondientes.
- ***start_test***: por cada set de particiones realizadas, este módulo se encarga de entrar a cada directorio e iniciar el test de carga para cada período dado.
- ***load_test***: este módulo implementa el test de carga, el cual, a partir de los archivos de configuración y la imagen de *Singularity* construida para las instancias de SUMO, ejecuta las simulaciones llamando al módulo encargado de paralelizar los procesos de simulación según el número de particiones realizadas. Como en el test para la versión secuencial, cada simulación se ejecuta un total de cincuenta veces, para luego promediar los tiempos de cada iteración.

Capítulo 6

Resultados

A continuación, se muestran los resultados obtenidos para el test de carga de la versión paralelizada de las simulaciones implementadas en SUMO:

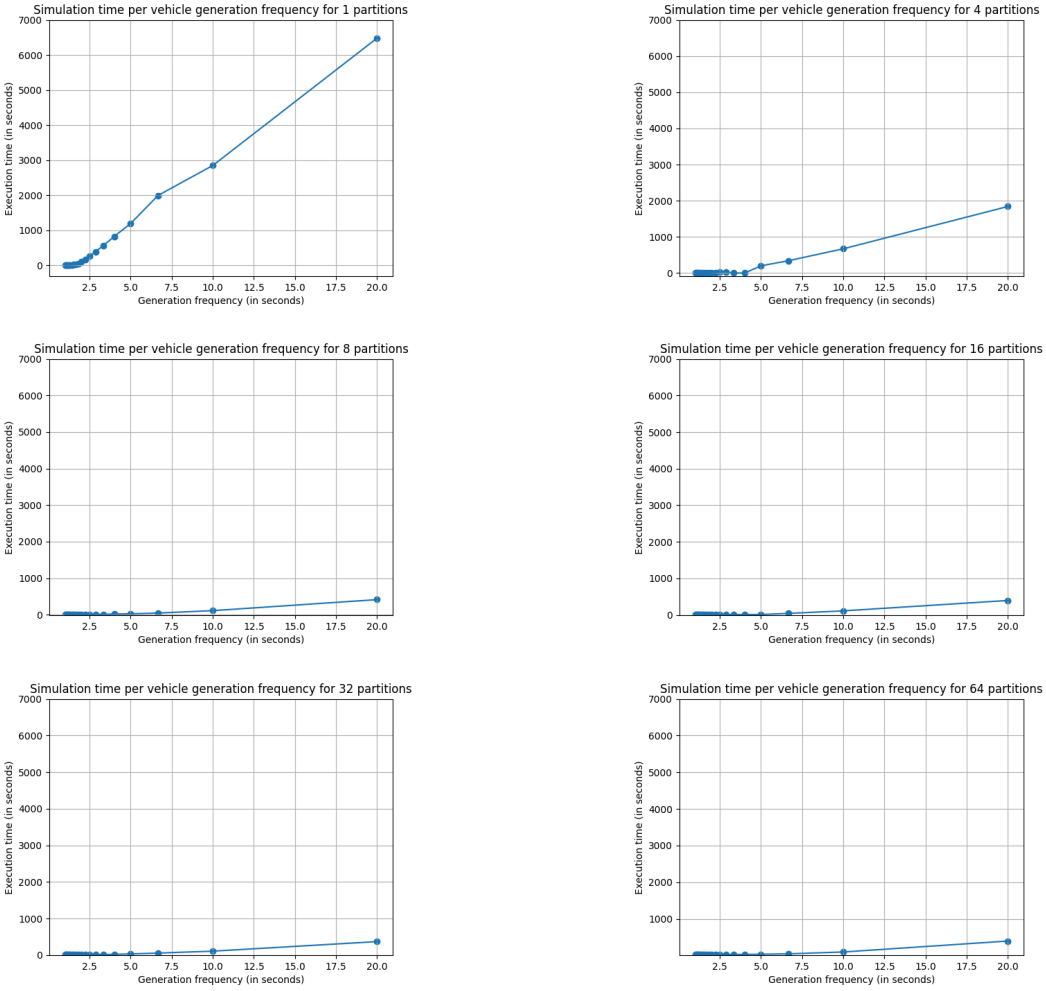


Figura 9: Resultados para la prueba de escalabilidad basada en la carga de tráfico vehicular para la versión paralelizada de SUMO.

Asímismo, en la siguiente figura se puede apreciar la comparación de la escalabilidad entre las simulaciones ejecutadas con distintos números de particiones:

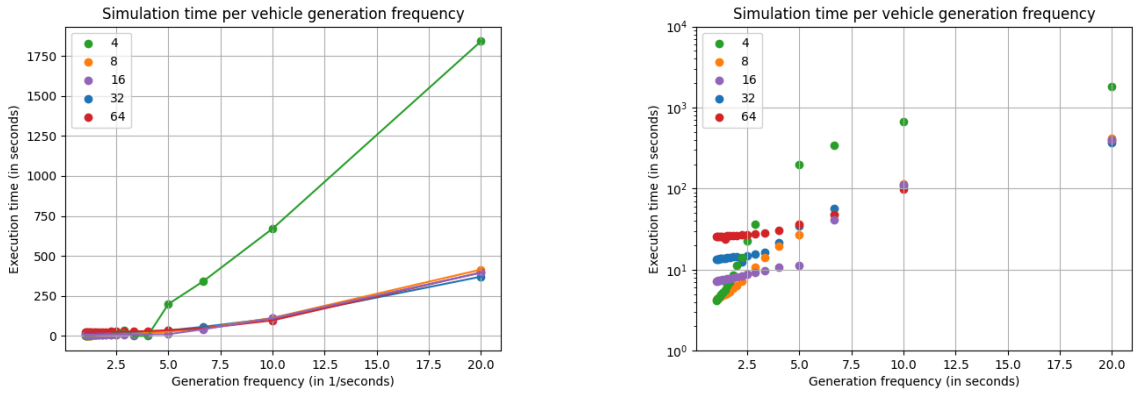


Figura 10: Comparación de escalabilidad de simulaciones ejecutadas con diferentes números de particiones.

Por otro lado, es posible observar el contraste en el escalamiento con la versión secuencial (es decir, una simulación de una sola partición) en el siguiente gráfico:

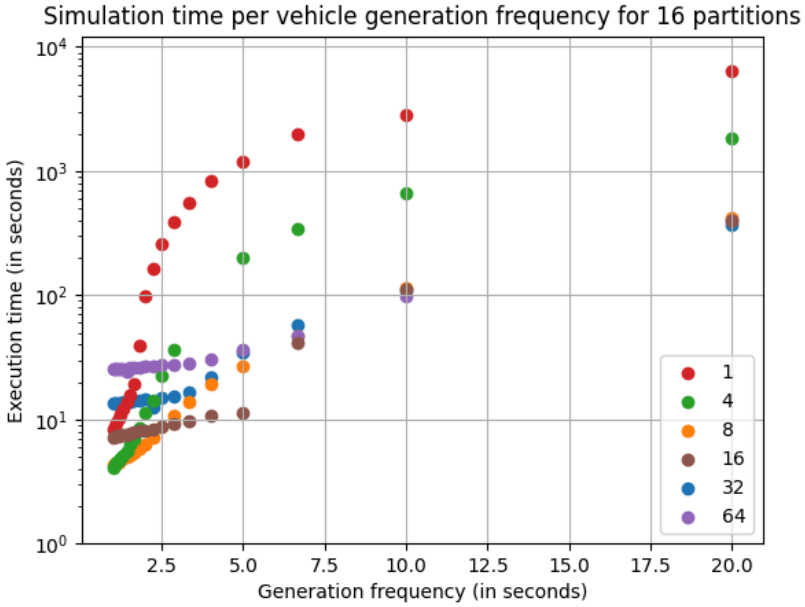


Figura 11: Comparación de escalabilidad de simulaciones con su versión secuencial.

En cuanto al uso de CPU para cada set de simulaciones, se obtiene el siguiente gráfico comparativo entre la escalabilidad de cada número de particiones realizadas:

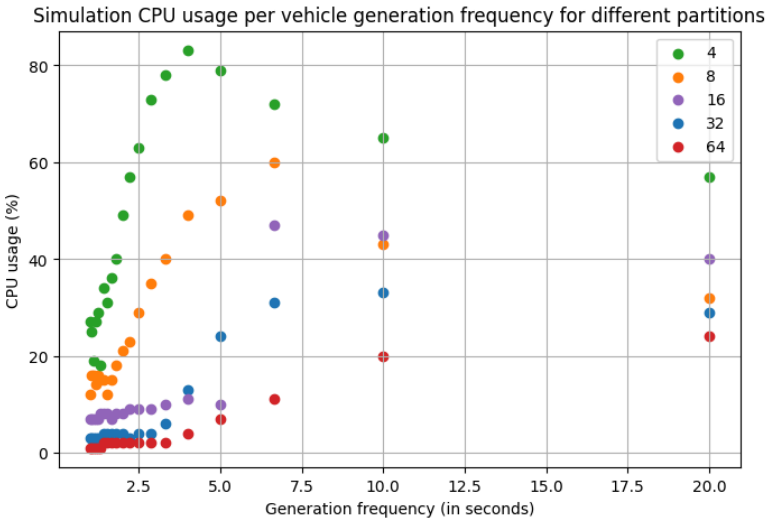


Figura 12: Comparación de escalabilidad de uso de CPU por simulaciones ejecutadas con diferentes números de particiones.

Capítulo 7

Discusión

Como es posible observar de la Figura 9, existe una gran diferencia entre la escalabilidad de la versión secuencial de una simulación y las versiones paralelizadas, teniendo estas últimas una cota superior considerablemente inferior a la cota superior de tiempo de las simulaciones no paralelizadas, lo cual muestra que las versiones paralelas de las simulaciones son más eficientes en cuanto al tiempo de ejecución de éstas.

Adicionalmente, es posible extraer de la Figura 10 que, a medida que se aumenta la cantidad de particiones, el escalamiento de las simulaciones aparenta ser cada vez más lineal, implicando también un mayor costo para las simulaciones con menor tasa de inserción de vehículos para simulaciones de grandes cantidades de particiones, pero un menor crecimiento en cuanto a los tiempos de ejecución de las simulaciones.

Cabe considerar en estos resultados dos aspectos importantes: el balance de las cargas en las particiones y la pérdida de información al realizar el corte de rutas.

Por un lado, el algoritmo aplicado para el particionamiento de los grafos correspondientes a la representación de los mapas conlleva un desbalance entre la cantidad de nodos presentes en las particiones que es importante a considerar, ya que existen particiones que superan considerablemente a otras en este aspecto; esto puede provocar que la escalabilidad de la solución se vea sujeta principalmente a la escalabilidad que ofrezcan los nodos con mayor carga, en vez de cada una de las particiones realizadas.

Por otro lado, el proceso de recorte de rutas según las particiones realizadas a los mapas implica una pérdida de información necesaria a considerar, dado que implica una menor carga para los nodos de ejecución, al eliminar vehículos cuyas rutas no cumplen con los criterios definidos por el algoritmo de particionamiento de rutas provisto por *cutRoutes.py*. Un posible trabajo a futuro consiste en revisitar este algoritmo para poder manejar de otra manera el corte de las rutas con los tiempos de partida de los vehículos y, de esta manera, evitar dicha pérdida de información.

En cuanto a los procesos de sincronización, se hace necesario implementar una manera de gestionar los tiempos de partida de los vehículos, de forma que no se produzca el fenómeno descrito en el apartado 5.6.3 y las rutas particionadas no queden como rutas independientes, sino que posean dependencia de unas con otras entre los nodos de ejecución. Dentro de la implementación también resulta necesario considerar el *overhead* que implican los procesos de comunicación entre los nodos, lo cual podría afectar de manera considerable a la escalabilidad de la solución ya implementada.

En cuanto al uso de CPU, es posible observar que hasta en las condiciones menos óptimas no se supera el 80% de uso, lo cual sugiere una mejora en cuanto al uso de recursos computacionales respecto a la solución secuencial, que ocupa, en el general de los casos, el 100% de la CPU que se le asigna para el proceso de simulación.

Capítulo 8

Conclusiones

Referencias

[1] Gligor Kotusevski y Ken A. Hawick, «A review of traffic simulation software», *Research Letters in the Information and Mathematical Sciences*, vol. 13, pp. 35-54.

[2] Paolo M. Ejercito, Kristine Gayle E. Nebrija, Rommel P. Feria, y Ligaya Leah Lara-Figueroa, «Traffic simulation software review», *IEEE*.

[3] Azise Oumar Diallo, Guillaume Lozenguez, Arnaud Doniec, y René Mandiau, «Comparative evaluation of road traffic simulators based on modeler's specifications An application to intermodal mobility behaviors», *SCITEPRESS - Science and Technology Publications*.

[4] Daniel Krajzewicz *et al.*, «The "Simulation of Urban MObility" package, An open source traffic simulation».

[5] Rainer Stark y Thomas Damerau, «Digital Twin», *Springer Berlin Heidelberg*. [En línea]. Disponible en: https://doi.org/10.1007/978-3-642-35950-7_16870-1

[6] Cheng Zhou *et al.*, «Digital Twin Network, Concepts and Reference Architecture». [En línea]. Disponible en: https://www.ietf.org/archive/id/draft-irtf-nmrg-network--digital-twin-arch-01.html

[7] Pablo López Álvarez *et al.*, «Microscopic traffic simulation using SUMO», *IEEE*.

[8] German Aerospace Center, «Introduction». [En línea]. Disponible en: https://sumo.dlr.de/docs/index.html.

[9] German Aerospace Center, «TraCI». [En línea]. Disponible en: https://sumo.dlr.de/docs/TraCI.html

[10] Lara Codeca, Raphaël Frank, y Thomas Engel, «Luxembourg SUMO Traffic (LUST) scenario, 24 hours of mobility for vehicular networking research», *IEEE*.

[11] Laura Bieker, Daniel Krajzewicz, AntonioPio Morra, Carlo Michelacci, y Fabio Cartolano, «Traffic Simulation for all, a real world traffic scenario from the city of Bologna», *Springer*.

[12] Sergey Ivanov, Ksenia Nikolskaya, Gleb Radchenko, Leonid Sokolinsky, y Mikhail Zymbler, «Digital twin of city, concept overview», *IEEE*.

[13] Edward Glaessgen y David Stargel, «The digital twin paradigm for future NASA and US air force vehicles».

[14] Ehab Shahat, Chang T Hyun, y Chunho Yeom, «City digital twin potentials; A review and research agenda», *Sustainability*, vol. 13, p. 3386.

[15] Javier Argota Sánchez-Vaquerizo, «Getting Real; the Challenge of Building and Validating a Large-Scale Digital Twinf of Barcelona's Traffic with Empirical Data», *ISPRS International Journal of Geo-Information*, vol. 11, p. 24.

[16] Krešimir Kušić, Rene Schumann, y Edouard Ivanjko, «Building a motorway digital twin in SUMO; Real-time simulation of continuous data stream from traffic counters», *IEEE*.

[17] Mariana Azevedo *et al.*, *Optimizing Vehicle IoT Systems; SUMO-Digital Twin Performance Analysis*. en 2024 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0 & IoT). IEEE, pp. 204-209.

[18] Raymond Klefstad, Yue Zhang, Mingjie Lai, R. Jayakrishnan, y Riju Lavanya, «A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation», *IEEE*.

[19] Toyotaro Suzumura, Gavin McArdle, y Hiroki Kanezashi, «A high performance multi--modal traffic simulation platform and its case study with the Dublin city», *IEEE*.

[20] Daigo Umemoto y Nobuyasu Ito, «Large-scale parallel execution of urban-scale traffic simulation and its performance on K computer», *Journal of Computational Social Science*, vol. 2, pp. 97-101.

[21] Matthew Franchi *et al.*, «PAVSP, a parallel autonomous vehicle simulation pipeline on High-Performance Computing».

[22] Nicolas Arroyave, Andrés Acosta, Jorge Espinosa Oviedo, y Jairo Espinosa Oviedo, «A new strategy for synchronizing traffic flow on a distributed simulation using SUMO».

[23] Hao Chen *et al.*, «QarSUMO; A Parallel, Congestion-optimized Traffic Simulator», *ACM*.

[24] George Karypis y Vipin Kumar, «A Fast and High-Quality Multilevel Scheme for Partitioning Irregular Graphs», *SIAM Journal on Scientific Computing*, vol. 20, pp. 359-392.

[25] German Aerospace Center, «Duarouter». [En línea]. Disponible en: https://sumo.dlr.de/docs/duarouter.html

[26] German Aerospace Center, «Randomtrips». [En línea]. Disponible en: https://sumo.dlr.de/docs/Tools/Trip.html

[27] Sylabs Inc., «Introduction to Singularity». [En línea]. Disponible en: https://docs.sylabs.io/guides/3.5/user-guide/introduction.html

[28] Andrew J. Younge, Kevin Pedretti, Ryan E. Grant, y Ron Brightwell, «A Tale of Two Systems; Using Containers to Deploy HPC Applications on Supercomputers and Clouds», *IEEE*.

[29] OpenMP Architecture Review Board, *OpenMP 5.2 Reference Guide*.

[30] German Aerospace Center, «Libtraci». [En línea]. Disponible en: https://sumo.dlr.de/docs/Libtraci.html

[31] Anthony Ventresque *et al.*, «SPartSim ":" A Space Partitioning Guided by Road Network for Distributed Traffic Simulations».

[32] Andrés Acosta, Jairo Espinosa, y Jorge Espinosa, «Distributed Simulation in SUMO Revisited":" Strategies for Network Partitioning and Border Edges Management».