GPU Accelerated Simulation of Transport Systems

Peter Thomas Heywood

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

The University of Sheffield
Faculty of Engineering
Department of Computer Science

February 2021

Acknowledgements

I would first like to thank Dr Paul Richmond and Dr Steve Maddock for their support, guidance and perseverance as my academic supervisors.

Thank you to Rob, Mozhgan and other members of the Visual Computing research group and Research Software Engineering team for their assistance and for making the office an enjoyable place to be.

I would also like to thank my friends for making Sheffield such an enjoyable place to live and especially the kayakers for the many days spent on the water.

Most importantly I'd like to thank my family, including my sister Sarah and my parents for their unending support.

I am grateful to Ian Wright and David Swain of Atkins for access and insight into the SATURN macroscopic simulation tool and for providing access to the real-world road networks. I would also like to extend thanks to Jordi Casas and Mark Brackstone of Aimsun for providing guidance and access to Aimsun microscopic road network simulator.

This work was supported by a Department for Transport Transport Technology Research Innovation Grant (T-TRIG July 2016) "Accelerating Transport Microsimulation: Demonstrating the impact of future many core simulations" [1], and additional support through EPSRC fellowship "Accelerating Scientific Discovery with Accelerated Computing" (EP/N018869/1)[2].

[1] https://www.gov.uk/government/publications/transport-technology-research-innovation-grants-t-trig-funding-winners

[2] https://gow.epsrc.ukri.org/NGBOViewGrant.aspx?GrantRef=EP/N018869/1

Abstract

Computer modelling and simulation of road networks are a vital tool used to evaluate, design and manage road network infrastructure. Road network simulations are however computationally expensive, with simulation runtime imposing limits on the scale and quantity of simulations performed within a reasonable time frame. This thesis examines the appropriateness of many-core processing architectures (such as GPUs) for the acceleration of microscopic and macroscopic road network simulation, and the potential impact on the choice of modelling approach.

Fine-grained agent-based microscopic simulations of individual vehicles are parallelised using GPUs, achieving high performance through a novel graph-based communication strategy for data-parallel simulations. A minimal benchmark model and scalable road network are defined and used experimentally to evaluate performance compared to Aimsun, a commercial simulation tool for multi-core processors. Performance improvements of up to 67x are demonstrated for large scale simulations.

High-level macroscopic simulations model network flow rather than individual vehicles. Although less computationally demanding than microscopic models, simulation runtimes can still be significant, often due to the calculation of many shortest paths. A novel Many-Source Shortest Path (MSSP) algorithm is proposed to concurrently find multiple shortest paths through sparse transport networks using GPUs. This is embedded within a commercial multi-core CPU macroscopic simulation tool, SATURN, and the performance evaluated on large-scale real-world road networks, demonstrating assignment performance improvements of up to 8.6x when comparing multi-processor GPU and CPU implementations.

Finally, the impact of the performance improvements to both modelling techniques are evaluated using a common benchmark model and the relative improvements demonstrated by the benchmarking of each approach using different transport networks. These results suggest that GPUs will allow modellers to shift towards using finer-grained simulations for a broader range of modelling tasks.

Declaration

I, the author, confirm that the Thesis is my own work. I am aware of the University's Guidance on the Use of Unfair Means (www.sheffield.ac.uk/ssid/unfair-means). This work has not been previously been presented for an award at this, or any other, university.

Contents

Preface

Acknowledgements . . . i

Abstract . . . ii

Declaration . . . iii

Contents . . . iv

List of Tables . . . vii

List of Figures . . . x

List of Algorithms . . . xvii

1 Introduction **1**

1.1 Aims and Research Outline . . . 3

1.2 Contribution to Knowledge . . . 4

1.3 Publications . . . 5

1.4 Thesis Structure . . . 6

2 Related Work **7**

2.1 Transport Network Modelling and Simulation . . . 7

2.1.1 Complex System Modelling and Simulation . . . 7

2.1.2 Calibration, Validation and Verification of Computer Simulations . . . 9

2.1.3 Road Networks and Users . . . 11

2.1.4 Properties of Road Network Graphs . . . 12

2.1.5 Sparse Graph Data-structures . . . 13

2.1.6 Summary . . . 14

2.2 Parallel and Distributed Processing . . . 14

2.2.1 Parallel Processing Approaches and Paradigms . . . 15

2.2.2 Parallel Programming Models . . . 17

2.2.3 NVIDIA GPUs and the CUDA Programming Model . . . 20

2.2.4 Summary . . . 26

2.3 Microscopic Road Network Simulation . . . 26

2.3.1 Architecture of a Microscopic Road Network Model . . . 27

2.3.2 Road Network Agent Based Models and Properties . . . 28
2.3.3 Microscopic Road Network Simulation Software . . . 31
2.3.4 Agent Based Modelling Frameworks . . . 32
2.3.5 Summary . . . 34
2.4 Combined Macroscopic Road Network Assignment and Simulation . . . 34
2.4.1 Architecture of a Macroscopic Assignment and Simulation Model . . . 34
2.4.2 Shortest Path Algorithms for Road Network Assignment . . . 38
2.4.3 Macroscopic Road Network Simulation Software . . . 45
2.4.4 Summary . . . 46
2.5 Summary . . . 46

3 GPU Accelerated Microscopic Simulation 48

3.1 Introduction . . . 48
3.2 CPU-based Microsimulation . . . 49
3.2.1 Aimsun . . . 51
3.3 Simplified Aimsun Model . . . 51
3.4 Benchmark Network . . . 52
3.5 Benchmark Experiments . . . 54
3.5.1 Grid-Scale Experiment . . . 54
3.5.2 Input-Flow Experiment . . . 54
3.6 CPU Benchmark Results . . . 55
3.7 GPU Microsimulation . . . 57
3.7.1 FLAME GPU . . . 57
3.7.2 FLAME GPU Implementation of the Gipps' Car Following model . . . 59
3.7.3 FLAME GPU Implementation of the Simplified Aimsun Model . . . 60
3.7.4 Cross Validation . . . 61
3.7.5 GPU Benchmark Results . . . 64
3.8 Summary . . . 67

4 Network-Based Communication for Data-Parallel ABM 73

4.1 Introduction . . . 73
4.2 Agent Communication in FLAME GPU . . . 74
4.3 Graph Based Communication . . . 75
4.4 Abstract Graph Communication Benchmark Model . . . 77
4.4.1 Benchmark Results . . . 80
4.5 Application to Simplified Aimsun Model . . . 82
4.5.1 Grid-Scale Experiment Results . . . 82
4.5.2 Input-Flow Experiment Results . . . 85

4.6 Summary 88

5 GPU Accelerated Macroscopic Assignment and Simulation 92

5.1 Introduction 92
5.2 SATURN 94
5.2.1 Shortest Path Calculations 96
5.2.2 Flow Accumulation 98
5.3 Real-World Benchmark Models 98
5.4 GPU Many Source Shortest Path for Sparse Graphs 99
5.5 GPU Flow Accumulation 104
5.6 Multi-GPU Implementation 108
5.7 Validation 109
5.8 Benchmark Results 110
5.8.1 Performance Compared to the Serial Baseline 115
5.8.2 Single-Processor Performance 116
5.8.3 Multi-Processor Performance 117
5.8.4 Multi-GPU Scaling 117
5.9 Summary 120

6 Comparison of GPU Accelerated Road Network Simulation Approaches 123

6.1 Introduction 123
6.2 Benchmark Network 124
6.3 SATURN Grid Network Benchmarking 127
6.3.1 Benchmark Model Parameters 128
6.3.2 Macroscopic Assignment and Simulation Results 128
6.4 Performance Comparison of Microscopic and Macroscopic Simulations 132
6.5 Summary 136

7 Conclusions 138

7.1 Summary of Main Findings 141
7.2 Future Work 141

Acronyms 144

Bibliography 148

List of Tables

3.1 Models selected for the Simplified Aimsun Model from the partial list of aimsun models and features. Aimsun publications and Documentation provide detailed descriptions of these functionalities ([143], [144], [243]) 52
3.2 Parameters for generation of the procedurally generated Manhattan-style grid road network 53
3.3 Network and Model parameters used for the Grid-Scale microscopic experiments. Parameters such as the time step, reaction time and detector period were selected as values commonly used within microscopic modelling in the UK as informed by the Aimsun developers. 54
3.4 Network and Model parameters used for the Input-Flow microscopic experiments. Parameters such as the time step, reaction time and detector period were selected as values commonly used within microscopic modelling in the UK as informed by the Aimsun developers. Multiple grid sizes were used to assess the performance impact of the input flow parameter at multiple network scales. 55
3.5 Details of the networks used for cross-validation. 62
3.6 Vehicle model parameters used for the deterministic cross-validation of the FLAME GPU and Aimsun models. 62
3.7 Truncated normal distributions for vehicle parameters used for the stochastic cross-validation of the FLAME GPU and Aimsun models. 63
3.8 Key data for the Constant Entrance Flow validation models 63
3.9 Key data for the Car Following Behaviour validation models 63
3.10 Velocity for the first vehicle in the 100m, 2250 vehicles per hour input flow, Car Following Behaviour validation model. 64
3.11 Key data for the Turing Proportion validation model. 64
3.12 Statistical summary data for the Deterministic Grid validation simulations. . . . 64
3.13 Statistical summary data for the Stochastic Grid validation simulations. 64

5.1 Serial Fortran per-routine run-time performance for large real-world benchmark model (5194 zones) sorted by time. The six longest-running routines are shown but with the mangled subroutine names removed. The most time consuming subroutine A is a key subroutine called within the assignment phase of the assignment simulation loop. This subroutine finds the flow of vehicles per edge in the network, based on the origin-destination information and the state of the network from the previous assignment-simulation loop. 96
5.2 Properties of the real-world SATURN networks used during development and benchmarking. The number of vertices and edges are shown for both the original and Spider (contraction hierarchy) variants. 99
5.3 Validation metrics for the multi-core CPU and GPU implementations of the LoHAM large-scale model. 110
5.4 The hardware used to benchmark real-world performance. The FP64:FP32 Ratio column of the GPU properties describes the ratio of double precision floating point (FP64) units within the GPU compared to single precision floating point(FP32) units. Most GPU architectures include a relative low number of FP64 units, as they are often not used for computer graphics applications. For General Purpose Computing on Graphics Processing Units (GPGPU) codes which may make heavy use of FP64 operations, including the flow-accumulation phase of SATALL, choosing a Graphics Processing Unit (GPU) architecture with a higher ratio of FP64 units can have a significant impact on performance. 111
5.5 Average runtimes in seconds for the total and assignment portions of SATALL application to complete across the three real-world road networks used for benchmarking, across a range of hardware. Note: Serial results are from a single run rather than an average due to runtime. 111
5.6 Relative speed-up compared to the single-CPU single-core results for single-socket CPU and single-GPU. Note: Serial results are from a single run rather than an average due to runtime. 115
5.7 Relative speed-up compared to the single-socket multi-core Central Processing Unit (CPU) results (system `P`). Higher is better. 116
5.8 Relative speed-up compared to multi-socket CPU (DC) 118
5.9 Relative speed-up compared to equivalent single-GPU results. 119

6.1 Parameters used for macroscopic simulations benchmarks of the artificial road network. 128

6.2 The hardware used to benchmark CPU and GPU SATURN simulations for the grid-scale experiment using the Manhattan-style procedurally generated grid network. 129

6.3 Average runtime (seconds) for the total runtime, assignment runtime and simulation runtime of each grid scale benchmark for the serial CPU (i7 6850k), multi-core CPU (i7 6850k using 12 threads), single GPU and dual GPUs (NVIDIA Titan V) macroscopic road network simulation implementation (SATURN). . . . 131

6.4 Relative performance improvement comparing simulator performance for a single Titan V GPU against a multi-core CPU implementation executed on an i7-6850k, for each timed phase of SATURN for the grid-scale macroscopic experiment . . . 132

List of Figures

2.1 Road network user mode distribution for a section of road in England (M1 North, J31 to J32) over the 1 hour period between 17:00 and 18:00 on the 6th May 2013 (GB Road Traffic Counts data set [65]). 11

2.2 The correlation between edges, vertices and zones for a small section of a one-way road network. 12

2.3 An simple weighted directed graph and the associated representation of the sparse adjacency matrix in COO, Compressed Sparse Row (CSR), and Compressed Sparse Column (CSC) formats. The graph contains 4 vertices and 6 weighted edges. Vertices are indexed from 0 to 3, with edge weights in the range 4 to 9. Edges weights are uniquely coloured to aid readability. 14

2.4 An illustration of the CUDA thread hierarchy, for a 2D grid of 6 blocks, where each 2D block contains 8 threads. The arrangement of threads within a 2D block is shown for block `(0, 1)`. 22

2.5 An illustration showing how the CUDA thread hierarchy allows a kernel using a grid of 6 thread-blocks to be executed on GPUs with 2 or 4 Streaming Multiprocessors (SMs). 23

2.6 An illustration of divergence occurring within an SM following the Single-Instruction-Multiple-Thread (SIMT) GPU architecture. Threads with odd indices perform separate operations to those with even indices, and therefore must be executed independently, prior to re-converging for a warp-wide operation. GPU Architectures which support independent thread scheduling will not necessarily reconverge without explicit synchronisation. 24

2.7 An illustration of a section of road network containing 3 sections of one-way road, two of which have vehicle class restrictions. The road network graphs representing these sections of road are shown for two vehicle classes, with edge weights representing travel time along the section. Due to the presence of the bus lanes, the road network graphs have differing values for each vehicle class. . . 36

2.8 A small directed weighted graph containing 9 vertices and and 10 weighted edges. 40

3.1 The average simulation run-time to complete a one hour simulation a total demand of 64000 vehicles in Aimsun 8.1, for different processor thread counts. The average of three simulations is shown. Error bars show the first standard deviation of the application runtime, but for some data points are smaller than the marker size. As processor thread count increases, total simulation time decreases, but with diminishing returns. Note that the Intel Xeon E5-2643 is a dual-socket system. No performance improvements were observed when using more processing threads than physical cores (Hyper-threading). 50
3.2 A 5×5 example of the procedurally-generated artificial grid network, showing the overall structure of the network and the arrangement of turning sections within a junction. The network can be scaled to any size, with networks of up to 576×576 used during benchmarking. 53
3.3 Total simulation time for Aimsun 8.1 as the total scale of the simulation is increased. Values shown are the average from 3 repetitions. The model and network parameters used for this benchmark are shown in Table 3.3. 56
3.4 Total simulation time for Aimsun 8.1 against input flow for procedurally generated road networks of grid size 64, 128 and 256. Run times shown are the average from 3 repetitions. The model and network parameters used for this benchmark are shown in Table 3.4. Transport for London guidance suggests maximum design capacities of between 900 and 1600 vehicles per hour per lane for urban, single carriageway, one-way roads with speed limits of 30mph ($\sim$ 50kmph) [244]. Benchmarks with input flows greater than 1000 vehicles per hour were not performed due to stop-sign induced queues preventing vehicles from entering the simulated region of the artificial benchmark road network which occurred more frequently as input flows were increased. 57
3.5 Per-Iteration simulation time for a model with grid size 128 and input flow 500 vehicles per hour. 58
3.6 Average iteration execution time for fixed grid of size $N = 16$ against agent population size, averaged over 100 iterations. Results generated using an NVIDIA Tesla K20c GPU. 68
3.7 Average iteration execution time per agent for fixed grid of size $N = 16$ against agent population size, averaged over 100 iterations. Results generated using an NVIDIA Tesla K20c GPU. 68

3.8 Flexible Large-scale Agent Modelling Environment for Graphics Processing Unit (FLAME GPU) per-iteration state diagram for *vehicle* and *detector* agents. Illustrates the logical control-flow of the agents within a single iteration of the simulation, arranged in simulation layers. Circles represent a type of agent in a given state, as described by the key. Agent functions are shown by rectangles, which can result in a change of state. Message lists are shown in green, with dotted arrows indicating the output and input of messages. 69
3.9 Car Following Model Velocity validation. The velocity of the first vehicle in the 100m Car following behaviour validation model with input flow set to 2250 vehicle per hour were manually extracted from Aimsun and FLAME GPU. Comparable results are shown. 70
3.10 Total simulation performance as the total scale of the simulation is increased, for each simulator. Values shown are the average from 3 repetitions. Executed using alternate communication strategies available in FLAME GPU 1.4. 70
3.11 Total simulation time against input flow for a procedurally generated road network of grid size 64. Run times shown are the average from 3 repetitions, using a logarithmic scale. The CPU simulation results from Aimsun 8.1 executed on a 4-core, 8-thread Intel i7 4770k are shown in purple. These are compared to results from the GPU simulations using all-to-all and spatially partitioned communication, shown by the orange and green series respectively, executed on an NVIDIA Titan V GPU. 71
3.12 Total simulation time against input flow for a procedurally generated road network of grid size 128. Run times shown are the average from 3 repetitions, using a logarithmic scale. The CPU simulation results from Aimsun 8.1 executed on a 4-core, 8-thread Intel i7 4770k are shown in purple. These are compared to results from the GPU simulations using all-to-all and spatially partitioned communication, shown by the orange and green series respectively, executed on an NVIDIA Titan V GPU. 71
3.13 Total simulation time against input flow for a procedurally generated road network of grid size 256. Run times shown are the average from 3 repetitions, using a logarithmic scale. The CPU simulation results from Aimsun 8.1 executed on a 4-core, 8-thread Intel i7 4770k are shown in purple. These are compared to results from the GPU simulations using all-to-all and spatially partitioned communication, shown by the orange and green series respectively, executed on an NVIDIA Titan V GPU. 72

3.14 Per-Iteration simulation time for a model with grid size 128 and input flow 500 vehicles per hour. Executed using Aimsun 8.1 (purple) on an i7 4770k, and using the FLAME GPU implementation with Spatially Partitioned Communication, on an NVIDIA Titan V GPU. 72

4.1 An example visually representing the data structure construction for the message processing algorithm. A simple directed graph is shown, containing 4 edges and 4 vertices, coupled with the equivalent CSR representation of the graph. 8 example agents represented by coloured triangles are shown on the road network represented by this figure, and the order of messages as output by these agents. The sorted order is then shown, coupled with the histogram, which provides access to the appropriate set of messages for a given target edge. 76

4.2 The FLAME GPU state diagram for the restricted flow graph model. It shows control flow within a single iteration of the simulation for the agents which only have a single state. Agent functions are represented by black rectangles, with green parallelograms representing message lists. The state diagram shows the 4 agent functions, arranged in layers which are executed sequentially. 78

4.3 A 4 row, 4 column example of the 2D grid of "pipes" used for the abstract graph communication benchmark model. Each vertex (grey circle) is connected to the neighbouring vertices in each cardinal direction, with a separate edge (arrow) in each direction, as properties such as capacity could be different in each direction. The benchmark experiments used a 32 by 32 grid. 80

4.4 Benchmark results for the Restricted Flow Graph model, for the two existing FLAME GPU communication strategies (Brute-force and Spatially Partitioned) and the proposed graph-based communication strategies, on a fixed size graph with varied populations. Results were collected using a NVIDIA Titan V GPU. The average of 3 runs is presented in seconds. This benchmark model has only been implemented for GPUs. 81

4.5 The effects of alternate FLAME GPU communication strategies on car following models is shown for a section of a grid-based road network, for a single agent represented in white. All-to-all communication results in 42 messages being parsed by the single agent. Spatially partitioned messaging results in the 18 messages from the agents in the blue shaded region, while the graph-based communication strategy results in only 5 messages from the orange bordered agents being processed. The spatially partitioned radius used for this illustration would be insufficient for accurate modelling, and is only used for illustrative purposes. . . 83

4.6 Total simulation performance as the total scale of the simulation is increased, showing the performance of the CPU simulations, and the alternate GPU simulations using a single GPU. Values shown are the average from 3 repetitions. A logarithmic scale is used to improve visibility of similar total simulation times. . 84
4.7 Total simulation performance as the total scale of the simulation is increased, for each simulator. Values shown are the average from 3 repetitions, for the CPU simulator and for the graph-based messaging GPU simulator using multiple GPUs. A logarithmic scale is used to improve visibility of similar total simulation times. 85
4.8 Total simulation time against input flow for a procedurally generated road network of grid size 64. Run times shown are the average from 3 repetitions. A logarithmic scale is used for the y axis. The results shown include the reference CPU simulator, Aimsun 8.1, executed on an i7-4770k (purple); and the results from FLAME GPU executed on an NVIDIA Titan V GPU using all-to-all (orange), spatial (green) and graph-based (pink) communication strategies. 87
4.9 Total simulation time against input flow for a procedurally generated road network of grid size 128. Run times shown are the average from 3 repetitions. A logarithmic scale is used for the y axis. The results shown include the reference CPU simulator, Aimsun 8.1, executed on an i7-4770k (purple); and the results from FLAME GPU executed on an NVIDIA Titan V GPU using all-to-all (orange), spatial (green) and graph-based (pink) communication strategies. 88
4.10 Total simulation time against input flow for a procedurally generated road network of grid size 256. Run times shown are the average from 3 repetitions. A logarithmic scale is used for the y axis. The results shown include the reference CPU simulator, Aimsun 8.1, executed on an i7-4770k (purple); and the results from FLAME GPU executed on an NVIDIA Titan V GPU using all-to-all (orange), spatial (green) and graph-based (pink) communication strategies. 89
4.11 Per-Iteration simulation time for a model with grid size 128 and input flow 500 vehicles per hour. Executed using Aimsun 8.1 (purple) on an i7 4770k and using the FLAME GPU implementations with Spatially Partitioned communication (green) and Graph-based communication (pink) on an NVIDIA Titan V GPU. . 90
4.12 The impact of FLAME GPU communication specialisation on the performance of the Car Following Model implementation can be separated into the cost of the message output and the message iteration within the car following agent function. FLAME GPU was modified to accurately record the cost of each process, and the average of three repetitions at each scale is shown. The simulations were executed on an NVIDIA TITAN X (Pascal) GPU. 91

5.1 The assignment-simulation loop within SATURN [235] 95
5.2 A simple example graph to illustrate the use of a vertex-frontier. 101
5.3 The contents of the vertex-frontier for the Bellman-Ford algorithm, for origin vertex `a` of the example graph in Figure 5.2 101
5.4 The contents of the origin-vertex-frontier within the the Many Source Shortest Path (MSSP) algorithm, for origin vertices `a` and `b` of the example graph in Figure 5.2 102
5.5 The size of the vertex-frontier and origin-vertex-frontier at each iteration of the modified Bellman-Ford algorithm for the LoHAM large road network, containing 5194 zones. The frontier size is shown for a single origin vertex and for all 5194 origin vertices. 102
5.6 The load-balancing effect of the cooperative thread array for 28 edges distributed across 8 threads. The imbalance of work has been minimised from a maximum divergence of 7 edges to a single edge. 103
5.7 The runtime (seconds) of the assignment phase of SATALL for the three real world models on various CPUs and GPUs. Lower times are better. Note that the Y axis is not shared between models. 112
5.8 The relative speed-up of the assignment phase of SATALL for three real world models compared to serial, single CPU core results. Higher is better. 116
5.9 The relative speed-up of the assignment phase of SATALL for three real world models on single CPU and single-GPU systems compared to a single multi-core CPU. 117
5.10 The runtime of the assignment phase of SATALL for three real world models comparing multi-socket CPU and multi-GPU implementations. 118
5.11 The relative speed-up of the assignment phase of SATALL for three real world models on multi-GPU systems, compared to multi-socket CPU results. 119
5.12 The relative speed-up of the assignment phase of SATALL for three real world models on multi-GPU systems, compared to the matching single-GPU system . . 120

6.1 A 5×5 example of the procedurally-generated artificial grid network used for microscopic simulations, showing the overall structure of the network and the arrangement of turning sections within a junction. The network can be scaled to any size, with networks of up to 576×576 used during benchmarking. 125

6.2 A 5×5 example of the procedurally-generated artificial grid network used for macroscopic simulations, showing the overall structure of the network including zones (pink squares), vertices (green circles), road sections (black arrows) and the arrangement of turning sections within a junction (orange arrows) including. The additional “flare lanes” are also shown. The network can be scaled to any size, with networks of up to 122×122 used during benchmarking. 127
6.3 The average total runtime of SATURN for the grid-scale benchmark, for CPU and GPU implementations. Values shown are the average from 3 repetitions. . . 130
6.4 The average runtimes of the assignment portion of SATURN for the grid-scale benchmark, for CPU and GPU implementations. The simulation portion is always executed in serial on the CPU. Values shown are the average from 3 repetitions. 132
6.5 The average runtimes of the simulation portion of SATURN for the grid-scale benchmark, for CPU and GPU implementations. The simulation portion is always executed in serial on the CPU. Values shown are the average from 3 repetitions. 133
6.6 CPU and GPU performance data against grid size for both microscopic and macroscopic simulations, for both CPU and GPU implementations. Figure 6.6a contains the total runtime for both microscopic and macroscopic approaches. Figure 6.6b shows the total microscopic runtime and the assignment portion of the macroscopic simulators, as this is the section of the application which uses the GPU and MSSP algorithm. 135
6.7 GPU implementation speedup relative to the respective CPU implementation for microscopic and macroscopic implementations. 136

List of Algorithms

1 Dijkstra's SSSP Algorithm 40

2 D'Esopo-Pape Sequential SSSP Algorithm 42

3 Original Bellman-Ford Single Source Shortest Path (SSSP) Algorithm 44

4 Existing Serial SATALL algorithm 97

5 Sequential Flow Accumulation 98

6 Many Source Shortest Path (MSSP) algorithm based on the Bellman-Ford algorithm, using an Origin-Vertex Frontier for execution on Single-Instruction-Multiple-Thread (SIMT) architectures. 105

7 GPU Flow Accumulation using double-precision atomic addition. This is suitable for Pascal-generation Nvidia GPUs and newer. 106

8 GPU accelerated flow accumulation algorithm using a reduced number of double-precision atomic operations. This is suitable for hardware with poor double-precision atomic addition performance, such as Maxwell and Kepler generation Nvidia GPUs. 107

Chapter 1

Introduction

Demand on transportation systems is increasing around the globe, for instance in the UK the Department for Transport (DfT) have projected increases of up to 42% in car ownership and up to 55% growth in UK road traffic demand between 2010 and 2040 [1]. Along with the increasing demand, existing network infrastructure is often underutilised resulting in reduced capacity available to transport network users [2].

The effects of increased demand and reduced capacity due to underutilisation can be countered through improved traffic management systems and improved network design tools [3]. Computer models and simulations allow transport authorities to perform virtual trials of proposed alterations prior to real world changes, removing the risk and cost of real world trials [4]. They may also be used to study the impacts of road network factors such as public health or the environment [5], [6]. Low-latency simulators can be integrated into automatic traffic management systems, reducing the need for human input to manage complex scenarios [7], [8].

Transport system simulators can typically be classified into one of three categories based on the granularity of the modelling: *macroscopic*, *mesoscopic* & *microscopic*. Macroscopic simulations use a *top-down* approach to accurately model the flow of transport through a network, using abstract representation such as modelling traffic as fluids [9], [10]. Mesoscopic simulations model use an intermediate resolution, often modelling the behaviours of groups (platoons) of individuals [11]. Microscopic simulations (*bottom-up*) are fine-grained models concerned with the behaviors of individuals and their local interactions [12].

Traditionally, macroscopic simulations were predominately used for transport modelling due to lower computational complexity than the finer-grained methodologies [13]. However, as computational resources have increased over time, microscopic simulators are becoming more prominent due to the benefits they provide, although the effectiveness and uptake of microscopic simulators is still limited by performance [14]. Microsimulations and Agent Based Modelling (ABM), a technique for naturally describing individual (agent) behaviours in a simulation [15], provide advantages over the more traditional macroscopic approaches for modelling transport

systems, such as improved capability for accurately modelling congested systems where the position of individual vehicles is relevant.

The runtime of road network simulations can be considerable, leading to sacrifices being made in the number of scenarios being considered and subsequently simulated. A frequently used solution is to consider only a single "average day" of the year, which may not accurately reflect any individual day [16]. A suitable approach to improve performance of transport network simulations is to parallelise the simulation, executing instructions concurrently on parallel hardware to yield higher performance and reduced runtimes. Macroscopic, mesoscopic and microscopic simulations are all suitable for parallel processing to varying degrees. Although transport simulations are not *embarrassingly parallel*, modern many-core processing architectures such as Graphics Processing Units (GPUs) offer the potential for significant enhancements to simulation performance. GPUs are high-throughput co-processors, which use many relatively simple processing cores to provide high levels of concurrency for data-parallel applications, compared to modern multi-core Central Processing Unit (CPU) which provide much fewer cores with higher per-core performance.

Using GPUs for parallelisation offers many advantages, for example, within High Performance Computing (HPC) GPU based computers are dominating both the Top 500 [17] and Green 500 [18] as they are can provide a large amount of computational power with lower electrical power requirements than more traditional CPU based HPC solutions [19]. However, using GPUs to increase performance brings additional complexity to achieve the desired levels of performance. The programming model requires specialist knowledge which end-user modellers may not possess and considerations must be made during implementation to achieve high levels of performance. Ultimately, additional performance made available to simulations through General Purpose Computing on Graphics Processing Units (GPGPU) may enable larger-scale and more-complex real-time (or better than real-time) simulations for use in transport system planning, management and analysis. Additionally, if GPGPU can successfully improve the simulation performance of road network simulations, the relative performance improvements to each modelling approach may influence the choice of modelling approach used. The high computational costs of fine-grained microscopic simulations is a significant factor in use of macroscopic approaches for large-scale simulations. If the performance gap between these opposing approaches is reduced, there may be a shift away from coarse simulation models towards finer-grained models, which can present modellers with additional information, potentially leading to more well-informed decisions.

1.1 Aims and Research Outline

This thesis aims to investigate and advance the use of many-core GPUs for the acceleration of road network simulations, considering both fine-grained microscopic models and coarse macroscopic modelling approaches. GPUs are highly parallel, high-throughput devices which must be provided with sufficiently high volumes of work to process concurrently. Microscopic road network simulations have the potential to be well-suited for parallelisation on GPUs due to a number of factors in their design. They are computationally demanding simulations, which perform fine-grained simulations of individuals within complex systems which may contain millions of individual vehicles in the case of national-scale simulations. Microscopic simulations can be implemented using an Agent Based Model (ABM) approach, which has been demonstrated as suitable for parallelisation [20]–[22]. Agent-based simulations of transport networks can potentially involve huge numbers of agents if large, complex road networks are being simulated. As such, microsimulation through an ABM approach potentially exposes the high level of parallelism required to take advance of modern many core processing hardware. However, GPGPU parallelisation is non trivial in a lot of cases, and care must be taken to both ensure that the adoption of parallel algorithms does not fundamentally change the result of simulation. Road network microsimulation software used within industry and in literature predominately leverages multi-core CPUs to improve simulation performance compared to serial implementations. Some research has been published into the application of many-core architectures such as GPUs for road network simulations [23]–[26], however, the adoption of fine-grained data-parallelism and GPUs has been limited, with little impact on the transport modelling sector or widespread adoption within literature thus far.

Macroscopic road network simulations are traditionally used in favour of finer-grained modelling approaches due to the reduced computational cost, provided by the higher level of abstraction. However, large scale macroscopic simulations can still have long run-times, even using state of the art approaches from research adopted by commercial multi-core CPU simulation packages. Time constraints can reduce the effectiveness of simulations and as a result modelling practitioners are restricted in the quantity, variety and therefore scale of simulations which can be completed within a reasonable period of time [14]. The approach to parallelisation within macroscopic road network simulations is less obvious than in the microsimulation case and will require the use of significantly different algorithms and data structures to access high levels of performance from many-core processor architectures, compared to those in use for the current state of the art macroscopic road network simulations for CPUs.

The choice of modelling approach used by practitioners is a compromise which is heavily influenced by the performance of simulations. Although GPUs are potentially suitable for parallelising both modelling approaches, the different approaches may see different levels of perfor-

mance improvement compared to the current multi-core CPU implementations used within the transport modelling industry. This may influence the decision making process when modelling practitioners must decide which approach is the most suitable for use, providing the desired level of detail while also completing within a reasonable time-frame.

The research questions asked by this thesis are as follows:

- Can modern GPUs be used to provide efficient and scalable microscopic road network simulations which offer high levels of performance compared to more traditional CPU-based approaches?
  - What are the essential models required for a road network simulation which can be used to evaluate the performance of microscopic simulations?
  - How can simulator performance be evaluated for a range of simulation scales?
  - What algorithms and data structures can be used to enable the desired levels of performance and scalability?
- Are modern GPUs suitable for the acceleration of macroscopic road network simulations, to reduce the time required to simulate large-scale road networks?
  - Which parts of an Macroscopic model will benefit the most from many-core parallelisation?
  - What algorithms and data structures must be used within a macroscopic model to access the high levels of performance offered by the GPU?
- What is the impact of GPUs on the choice of modelling approach, which currently favours high level macroscopic modelling approaches rather than fine-grained microscopic models due to computational cost and long application runtimes?

1.2 Contribution to Knowledge

This thesis makes the following contributions:

- **C1** - A minimal subset of agent-based road network simulation behaviours is defined for a microscopic road network model, with an associated scalable artificial benchmark road network which supports the evaluation of road network simulation performance at a range of simulation scales.
- **C2** - A general-purpose graph-based communication strategy is presented for high performance agent communication for fine-grained data-parallel agent based simulations, which enables high performance agent based simulations of transport networks on GPUs.
- **C3** - A benchmark ABM is proposed and used to evaluate the performance impact of the general-purpose graph-based communication strategy for GPU accelerated ABMs, without other complexities of road network simulation.

- **C4** - The proposed microscopic road network simulation model is implemented for execution on GPUs, and the performance is evaluated against an equivalent model in a commercial multi-core CPU software tool, demonstrating improvements to simulation performance and performance scalability.
- **C5** - A novel algorithm (Many Source Shortest Path (MSSP)) is presented for concurrently finding the shortest path between multiple origin and destination vertices within a transport network. The algorithm is able to extend the current state of the art in path finding within sparse transport networks by solving a number of shortest path calculations concurrently when using data parallel accelerators such as GPUs.
- **C6** - The proposed many-source-shortest-path algorithm (C5) is embedded within a macroscopic road network simulation tool, SATURN, to evaluate the impact of such an approach on macroscopic road network assignment and simulation models. Performance advances are demonstrated compared to the existing multi-core implementation, for large-scale real-world road networks.
- **C7** - The relative impact of GPU acceleration on microscopic and macroscopic road network simulations is compared and evaluated through the shared benchmarking model.

1.3 Publications

The work carried out during the completion of this thesis has resulted in the following publications:

- P. Heywood, S. Maddock, R. Bradley, D. Swain, I. Wright, M. Mawson, G. Fletcher, R. Guichard, R. Himlin, and P. Richmond, "A data-parallel many-source shortest-path algorithm to accelerate macroscopic transport network assignment", Transportation Research Part C: Emerging Technologies, 2019 [27]. This publication is related to Contributions C5 and C6.
- P. Heywood, S. Maddock, J. Casas, D. Garcia, M. Brackstone, and P. Richmond, "Data-parallel agent-based microscopic road network simulation using graphics processing units", Simulation Modelling Practice and Theory, 2017 [28]. This provides the basis of Contributions C1, C2, C3 and C4.
- P. Heywood, P. Richmond, and S. Maddock, "Road network simulation using FLAME GPU", Euro-Par 2015: Parallel Processing Workshops, Springer, 2015 [29]. This publication contains early work related to Contributions C1 and C3.

1.4 Thesis Structure

- Chapter 2 (Related Work) presents general background information related to road network simulation and the use of computer parallelism including GPGPU, with a more focussed review of microscopic and macroscopic road network simulations.
- Chapter 3 (GPU Accelerated Microscopic Simulation) describes the application of GPUs to microscopic road network simulations through ABM, by defining a set of models and associated scalable road network. The method by which the GPU ABM can be implemented is described, and a performance evaluation made against a state of the art, commercial microsimulation software tool.
- Chapter 4 (Network-Based Communication for Data-Parallel ABM) extends and improves the performance of the GPU accelerated ABM described in Chapter 3 through the use of a specialised communication pattern. The general purpose graph-based communication pattern for GPU accelerated ABMs improves the communication-efficiency of road network behavioural models within agent based models on many-core processors, leading to significant performance improvements and scaling behaviour. This is complemented by an abstract ABM model designed to benchmark the communication patterns presented in road network behavioural models.
- Chapter 5 (GPU Accelerated Macroscopic Assignment and Simulation) presents the application of GPUs to macroscopic transport network simulation and assignment modelling, focussed on accelerating the performance of the computationally expensive assignment of traffic routes. This is achieved through the development of a novel shortest path algorithm which caters for graphs characteristic of transport networks. The novel algorithm is embedded within a commercial simulation tool, and the performance is evaluated against the serial and multi-core CPU versions of the simulator using a set of real-world models.
- Chapter 6 (Comparison of GPU Accelerated Road Network Simulation Approaches) evaluates the relative benefits of GPU acceleration on road network simulations using either microscopic or macroscopic techniques. This is achieved through the use of the scalable benchmark network defined in Chapter 3, but applied to the macroscopic assignment and simulation tool from Chapter 5.
- Chapter 7 (Conclusions) provides conclusions to the work carried out in this thesis, and provides suggestions for future work in this area.

Chapter 2

Related Work

This chapter provides context and background for the parallel computer simulations of transport networks. Section 2.1 introduces transport network modelling and simulation. Section 2.2 provides a discussion of parallel processing approaches with emphasis on data-parallelism for many-core processing architectures and Graphics Processing Units (GPUs). A more focussed review of literature and techniques used for microscopic road network simulations are provided in Section 2.3. Section 2.4 presents a similar review of related work for macroscopic modelling and simulation of road networks, including shortest path algorithms which are fundamental to this modelling approach. Finally, Section 2.5 summarises the chapter.

2.1 Transport Network Modelling and Simulation

Complex systems such as transport networks are modelled and simulated on computers in order to provide insight into the properties and characteristics of the target system and predict the effect of changes without the need for expensive real-world trials. For transport networks, simulations can be used to support both design and planning of changes to transport networks [30]–[32]. Transport simulations can also be used to support decision making within management of active transportation networks [8], [33], [34] and study the impacts of road networks on society and the environment [5], [6]. The following subsections discuss the general concepts and use of complex system simulations and how they related to the domain of transportation networks.

2.1.1 Complex System Modelling and Simulation

A computer *Simulation Model* is an abstract representation of a complex system which captures the behaviour of the target system at a (typically) simplified level, in an attempt to produce a representation of the system which is easier to understand [35]. The model must be implemented as a *Computer Simulation*, which can be executed one or more times in order to gain insight

about the modeled system. In some cases an individual simulation can provide meaningful insight into the modelled system, however, most complex system simulations involve a degree of stochasticity to capture variance within the modelled system. When this is the case, an *Ensemble* of many individual simulation runs must be executed, and aggregate operations performed over the ensemble to gain meaningful and representative results [36].

Models and simulations of complex systems can be categorised in many ways based on various properties of the model. Models which contain spatial elements can be classified as either *discrete-space* or *continuous-space* models. Discrete space simulations divide the simulation environment into a set of discrete locations to reduce complexity. Cellular Automata are one such example of discrete space models [37]. Continuous space simulations represent the environment as a single continuous space in one or more dimensions, allowing free movement within the environment. Models using continuous space for environments are often bound by fixed dimensions, or they may implement environment wrapping, forming a torus [38].

Similarly, the progression of a simulation can be modeled as discrete events or continuous time (where simulations progress iteratively at regular intervals). *Discrete-event simulations* model the system as a sequence of distinct events, each of which lead to a state of change in the system and are well-suited for modelling asynchronous systems where events occur at irregular intervals [39], [40]. In contrast, *Continuous time simulations* perform periodic iterative updates to the simulation state, which can use a fixed time-step. Event based simulations are typically more work-efficient than continuos time simulations, requiring fewer computational resources, as they only perform updates when required. Systems with frequent changes of state, such as the position of vehicles within a simulation, are more likely to follow a continuous time approach.

Complex systems can be modelled and simulated at differing levels of detail, which are often categorised as *Macroscopic*, *Mesoscopic* or *microscopic* models or simulations. Macroscopic models are the most coarsely grained approach, typically following a top-down equation-based approach [41]. Within transport system simulations this often involves the modelling of traffic flow on sections of road [42]–[44]. Mesoscopic models operate at a mid-level of granularity, covering a the spectrum of modelling resolutions which lie between the macroscopic and microscopic approaches. For transport networks models this can mean the modelling of groups or platoons of individuals as a single unit [11]. Microscopic models are fine-grained models following a bottom-up approach. Often modelling each individual in the system, their local interactions and the environment [12]. Agent Based Models (ABMs) are a form of microscopic model, which offer a natural mechanism for describing a system of behavioural entities [45]. The behaviours of individuals are modelled, and through the interaction between one another and the environment more complex higher-level behaviours can emerge. The specific meaning of each of the macroscopic, mesoscopic and microscopic terms varies between modelling domain, but the relationships between level of details are consistent.

Each modelling resolution has different advantages and disadvantages which influence which modelling approach is most appropriate for a given task. Fine-grained microscopic models are less abstract than higher level approaches such as macroscopic models. Through techniques such as ABM microscopic models can be more intuitive to develop [45] and more interpretable by non-modelling specialists who may be key stakeholders in any decision making processes than more abstract purely mathematical approaches. The abstract nature of high-level approaches can however be advantageous. Macroscopic models are not as data-intensive as mesoscopic or microscopic road network simulations. This can reduce the volume and complexity of data required for model calibration and validation, which can be very costly to gather [46]. It also leads to lower-resolution data being generated through the simulation approach, only providing information at an aggregate level, rather than being able to view the movements of individual vehicles and how they may interact with one another. Combined with simulation iterations representing longer periods of time this can lead to high-impact short-term events being missed, as the effects are aggregated over a longer time period [47]. One of the most significant effects of the choice of modelling approach is on simulation runtime. Macroscopic simulations typically run in a much shorter duration than finer-grained approaches of the same network, with greater differences in performance at larger scales of simulation. In addition, microscopic road network simulations are often highly stochastic, requiring large ensembles of simulations to produce statistically relevant results [36]. This exacerbates the long simulation runtime of the computationally expensive microscopic simulations.

Hybrid approaches can be used to combine aspects of the different modelling resolutions to balance the advantages of each approach. For instance, a higher-level macroscopic or mesoscopic model can be used to simulate a large transport network, with finer-grained microscopic modelling used for areas of the simulation where more detail is required due to complex interactions [11], [48]. Alternatively, the sub-components of a model may operate at different levels of granularity, using macroscopic approaches to assign transport demand, which are then simulated using a mesoscopic or microscopic approach [49].

2.1.2 Calibration, Validation and Verification of Computer Simulations

To ensure that computer simulations of a model are accurate they must be verified, calibrated and validated. A computer simulation can be *verified*, to ensure that the implemented computer simulation matches the theory, concepts and intent of the underlying mathematical computer model [50]. For a software implementation to be correct, software engineering best practices [51]–[53] should be followed to minimise the likelihood of errors [54]. Models of complex systems typically contain many model parameters which can be used to *calibrate* the model, where the parameters are adjusted so that the simulation outputs are comparable to observed or expected results within an acceptable degree of tolerance [55]. The calibration process requires data to

calibrate the system against. For the calibration to be successful the data should be broad in nature, for instance, containing data collected from both major and minor roads, or from both cities and the countryside such that the relevant parameters can be calibrated to suit the demands of the components they effect [56]. The collection of data for calibration purposes is often the most expensive and time consuming part of the calibration process, but the adoption of Intelligent Transport System (ITS) around the world has greatly increased the amount of data available [46].

Many methods can be used to calibrate computer models [57]–[59]. For instance, the simplex method represents the parameter space of the model using a geometric feature of $P+1$ vertices where P is the dimensionality of the parameter space [57]. The simulation is executed using each parameter set in the feature, and the fitness with respect to the calibration data is evaluated. The least-fit parameter set is discarded and replaced by a new feature set. Over many iterations this results in the parameter set moving towards the optimal solution, including when applied to road network simulation models [46]. An alternative approach for calibration could be to use machine learning approaches such as Genetic Algorithms (GAs). GAs are a heuristic search technique inspired by natural selection [60]. Sets of model parameters are represented by individuals within a population. Simulations are performed using the parameters encoded by each individual within the population and the fitness of each simulation is evaluated. A new population of model parameters are then generated, using biologically inspired processes such as mutation and crossover. The new population is then used to run a batch of simulations and generate new fitness values. This generational process is repeated many times, producing new populations which should be collectively fitter than the previous generation, ultimately converging on a more optimal solution [55], [58], [61].

Once calibrated, the model can be *validated*, where the validity of a computational model can be considered as the testability of the model [54], which can be split into six types of validation as defined by Knepell and Arangno: conceptual, internal, external, cross-model, data, and security [62]. Validation can also be considered to be the process of determining the degree to which a model represents the real world [50]. This process requires real world data to compare the output of the simulation to, and check if the results are within an acceptable range of the observed data. The data being used in order to validate a model needs to be independent from the data used to calibrate the model [63] in order for the validation to have any meaning. The traditional methods of collecting real-world data for calibration and validation came with high costs and limited information, but with increasing use of automatic techniques for capturing data (such as Smart Motorways) this availability of data is increasing.

2.1.3 Road Networks and Users

Within the context of modelling transport systems, the various classes of road users are an important consideration. Road networks are used by a wide range of users including pedestrians, cyclists, two-wheeled motor vehicles such as motorbikes, cars, buses, Light Goods Vehicles (LGVs), Heavy Goods Vehicles (HGVs) and more recently Connected and Autonomous Vehicless (CAVs). The ratio of different *modes* of transport varies depending on the type and the location of a road, depending on who is using the road, why the vehicles is being used, where the vehicle is heading and laws governing which vehicles are allowed on each type of road (in a given location). A typical example of the distribution of vehicles on UK motorways is shown in Figure 2.1 which shows that travel during the observed time of day is predominately Cars with some HGVs & LGVs, a few buses and Motorcycles but no pedal vehicles (which are not allowed on Motorways in the UK). The Highways Agency report that the most common mode of traffic in 2013 using the Strategic Road Network (SRN) based on the number of vehicle miles covered is the LGV; followed by cars, HGVs, motorcycles and buses [64]. This variance needs to be accounted for by road network models, to accurately capture the behaviour of the transport network, as the mix of modes of traffic will have varying degrees of impact on the transport network.

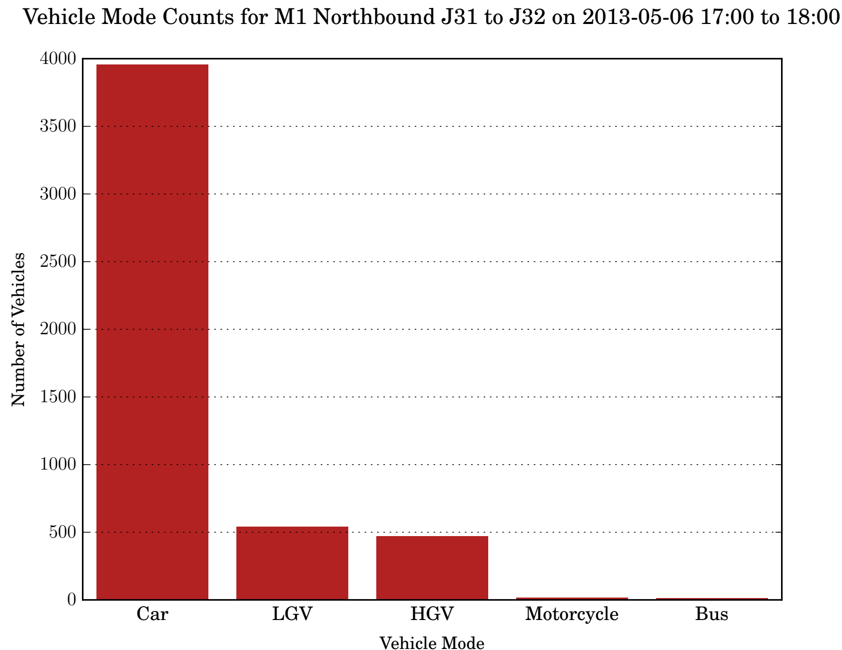


Figure 2.1: Road network user mode distribution for a section of road in England (M1 North, J31 to J32) over the 1 hour period between 17:00 and 18:00 on the 6th May 2013 (GB Road Traffic Counts data set [65]).

The road network on which vehicles travel is physically made up of sections of road connected by junctions. This naturally leads to the use of graphs to represent road networks within modelling and simulation. A directed graph (G) is comprised of a set of vertices or nodes (V)

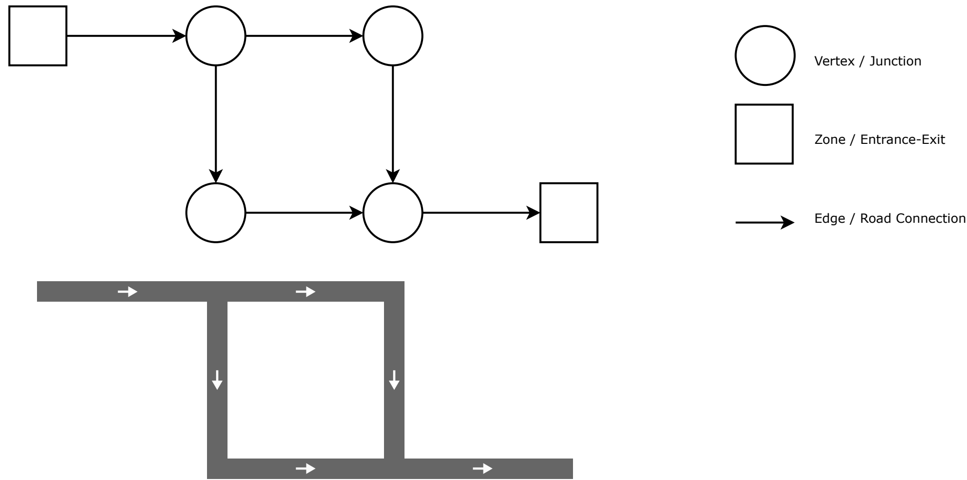


Figure 2.2: The correlation between edges, vertices and zones for a small section of a one-way road network.

and a set of one-way directed edges or links (E) connecting a pair of vertices. Edges may also have additional properties associated with them, typically a weight or cost (C) leading the the graph being referred to as a weighted graph. Within transport network simulations, edges may have many properties rather than a single weight, such as the number of lanes, or speed limit for that section of road. Vertices may also have information associated with them, such as the position within 3D space. In some road network applications the set of vertices V may contain multiple types of vertex. In addition to a set of vertices which represent intersections within the road network connecting links, a set of virtual vertices are often used to allow vehicles to enter and exit the simulated region, or be associated with origin-destination demand data. These are often referred to as *Zones* or *Centroids*. Figure 2.2 shows an example of how zones, vertices and edges may be used to represent a small section of a one-way road network.

2.1.4 Properties of Road Network Graphs

Simulation of networks can be achieved using a range of standard network algorithms for traversal and path calculations. Several characteristics of graphs are important when considering algorithm selection including: (i) *degree*; (ii) *density*; (iii) and *diameter*. The degree of a vertex is the number of edges directly connected to the vertex. Road network graphs typically have a very low average degree, due to the physical properties of road networks. The density of a graph describes the relationship between the number of edges and vertices with the graph, and is defined as $D = \frac{|E|}{|V|(|V|-1)}$ for directed graphs. It is the ratio of the number of edges in the graph compared to the number of edges in a fully connected graph [66]. A dense graph contains a large number of edges per vertex, with an upper bound of $\frac{|V|(|V|-1)}{2}$ for undirected graphs. The closer to the upper bound, the denser the graph. Sparse graphs in comparison have a low number of edges per vertex. Road networks are generally sparse, with low densities. The diam-

eter of a graph is the greatest distance (number of edges) between any vertex pair in the graph (considering shortest paths for weighted graphs). The diameter of a transport network graph is often relatively high. Sparsity, diameter and degree significantly differentiate road network graphs from other categories of graph, such as social media interaction graphs which tend to have low diameters with very high density, leading to alternate algorithm selection for similar tasks.

2.1.5 Sparse Graph Data-structures

Road networks represented as graphs can be stored in many different data structures when used in computer applications. The differing approaches typically each have their own advantages and disadvantages with respect to the time complexity (an estimation of how much time a computer algorithm will require based on the number of elementary operations required) [67] and performance of certain operations, such as the construction of the graph, addition or removal of edges or vertices, or the alignment of elements within memory. Properties of the graph also impact data structure selection, especially the density of the graph.

As road networks are generally represented by very sparse graphs, they are generally represented using sparse adjacency matrices. An *adjacency matrix* encodes the connectivity between vertex i and vertex j in the matrix element a_{ij}. For directed graphs the matrices are asymmetric. Weighted graphs will encode the weight within the matrix, whilst non-weighted graphs would just store a boolean value. The adjacency matrix of a sparse graph would contain many zero values, considerable memory can be saved using a sparse data structure, which only encodes non-zero values. There are many approaches for storing a sparse matrix, which are selected based on the intended use of the matrix. Some formats are efficient for creating or modifying the graph, such as the triplet or COO format. Other formats are more well suited to efficient access, such as Compressed Sparse Row (CSR) and Compressed Sparse Column (CSC).

Figure 2.3 demonstrates how a simple weighted directed graph containing 4 vertices and 6 edges can be represented in the COO, CSR and CSC formats. The triplet or COO format of graph representation is very efficient for construction, and converting to other data sparse matrix data structures, but is much less efficient than other formats when just accessing the data contained within the matrix. The COO format is made up of a list of tuples, containing the row, column and value of the matrix. For a sparse graph, this is the source vertex, origin vertex and edge value(s) for each edge. Ideally the edges are stored in order of row and column index to improve access performance however this is not required. The CSR format on the other hand has higher construction costs, but reduced access costs, which can be important when the data structure must be queried many times, but infrequently mutated. The sparse graph is stored using a set of three one dimensional arrays: the weight of the edge in `V`, the column index or destination vertex for each edge in `COLUMN_IDX`, and `ROW_IDX` contains the starting position

within V and COLUMN_IDX for each row or source vertex, with an additional value for the total number of edges in the graph. For directed graphs this format allows very efficient access when discovering links which leave a source vertex, for a given source vertex index u, ROW_IDX[u] provides the index of the first edge to leave the source vertex, and ROW_IDEX[u+1]-1 is the index of the last edge to leave the source vertex. The CSC format is very similar to the CSR format, but roles of ROW_IDX and COLUMN_IDX are reversed such that the incoming edges to a destination vertex can be efficiently discovered and iterated.

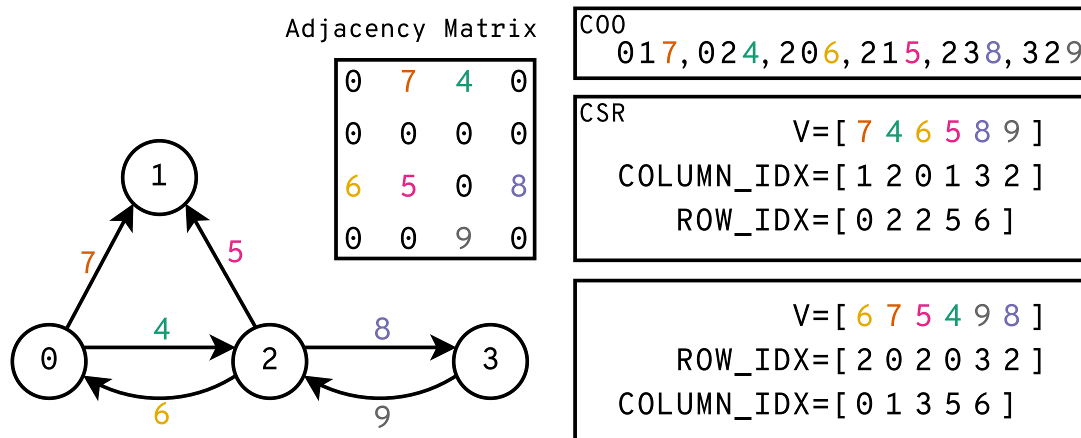


Figure 2.3: An simple weighted directed graph and the associated representation of the sparse adjacency matrix in COO, CSR, and CSC formats. The graph contains 4 vertices and 6 weighted edges. Vertices are indexed from 0 to 3, with edge weights in the range 4 to 9. Edges weights are uniquely coloured to aid readability.

2.1.6 Summary

This section has described the general properties and characteristics of computer modelling and simulation, the properties of road networks and the significance of different road users. Additionally this section has highlighted the importance of data structures used for road network representation. The choice of data structure has a significant impact on how data within the road network graph can be used during simulation. The data structure choice is also vitally important when considering parallel and distributed simulation. Section 2.2 explores how parallel and distributed computing can be more generally used to reduce the runtime of software applications, including computationally demanding simulations. The impact of this and the importance of data structures is later considered in Sections 2.3 and 2.4.

2.2 Parallel and Distributed Processing

Computationally demanding tasks such as simulation of transportation systems can have significant application runtimes. Parallel computing can be used to reduce the duration of time

required to gather the results of one or more simulations, by executing operations concurrently. This can be used to improve the performance of a single simulation or application (High Performance Computing (HPC)), or to reduce the total time to execute many instances of an simulation or application by improving throughput (High Throughput Computing (HTC)).

Historically, the performance of individual processors was improved over time through *Frequency Scaling*, performing operations at increasingly higher rates. Moore famously observed that the transistor density of processors was doubling roughly every two years [68]. When combined with Dennard Scaling [69], a scaling law which states that as transistors reduce in size their power density stays constant, the increasing transistor density over time enabled the performance per watt for processors to increase at an even greater rate. However, processor frequency is directly correlated with power usage ($Power = Capacitance \times Voltage^2 \times Frequency$) which combined with current leakage at smaller transistor sizes ultimately led to the end of frequency scaling as the primary method of reducing application runtime. Increased frequency can not be sustained within reasonable power and cooling constraints. Similarly, pipeline designs reached a limit on the complexity of transistor usage so instead, chip design moved towards an approach of utilising transistors to build multi core architectures with more or less fixed frequency and power requirements [70]. Utilisation of these parallel architectures requires fundamentally different approaches to achieve high performance.

When parallelising an application, the potential speedup (relative performance improvement) is influenced by several key factors. Applications cannot always be fully parallelised, with a portion of the application being executed in serial. Amdahl's law [71], [72] can be used to provide an upper limit on the potential speedup of an application based on the proportion of the application which cannot be parallelised. For instance, if only 90% of an application can be parallelised then the application latency can only be improved by a factor of 10, regardless of how many processors are used. Amdahl's law assumes that the problem size will remain the same regardless of the number of processors available. Gustafson proposed an alternative relationship which accounted for the size of the problem changing as number of processors increased, with the serial portion of the application often requiring the same time regardless of problem size, leading to less pessimistic approximations of potential speedup [73]. In either case highest performance is always obtained when an entire application is parallelised.

2.2.1 Parallel Processing Approaches and Paradigms

Parallel algorithms can be classified as either *task-parallel* or *data-parallel*. Task-parallelism is the decomposition of a task into independent sub-tasks which are executed concurrently by multiple processors or threads, on the same or different data. Typically each thread will execute the same or different operations on the relevant data, and communicate with other threads in order to complete the overall task [74]. Data-parallelism performs the same task concurrently over

the different elements of a data set, which can occur at differing levels of granularity depending on scale of the unit of data being processed concurrently [75]. Additionally, processors can often perform Instruction Level Parallelism (ILP), where multiple independent processor instructions can be overlapped within the same thread of execution. ILP is typically implemented at the hardware level (e.g. via the pipelining of instructions, allowing multiple instructions to be executions within a single clock cycle), or in software during compiler optimisation, rather than as an algorithmic decision [76].

Parallelism can be expressed at several levels within a computer system. Many independent processing nodes with local memory may be connected to one another through network connections, forming a distributed memory. Within a processing node, there may be multiple processors which can operate concurrently with access to shared local memory. Parallel systems may be of one common architecture (homogeneous systems), or they may contain multiple processor architectures forming a heterogeneous system. Heterogeneous systems leverage the advantages of multiple processing architectures such as Central Processing Units (CPUs), GPUs, Field-Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs) to provide advantages over more traditional homogeneous systems, although they may not be suitable for all tasks [77].

There are many different parallel computing paradigms for different classes of parallel processing hardware. Flynn's taxonomy of parallelism [78] offers a good categorisation of the alternate approaches, based on the number of instruction streams and number of data streams which the instructions operate over. Flynn originally proposed 4 categories, although these have since been extended:

- Single-Instruction-Single-Data (SISD) are sequential architectures which exhibit no parallelism;
- Single-Instruction-Multiple-Data (SIMD) architectures apply the same instructions to many elements of data in lockstep, exploiting data-parallelism;
- Multiple-Instruction-Single-Data (MISD) architectures are used in fault-tolerant systems;
- and Multiple-Instruction-Multiple-Data (MIMD) includes both shared-memory processors, where multiple processing units can directly access the same memory, and distributed memory systems, where it is usually not possible to directly access the memory which belongs to another processor.

Single-Instruction-Multiple-Thread (SIMT) architectures combine Flynn's SIMD classification with multi-threading and used within modern GPUs [79]. Rather than every instruction being applied to all data elements being processed in lockstep, the instruction may only apply to certain some of the data elements active within the processors.

Single-Program-Multiple-Data (SPMD) and Multiple-Program-Multiple-Data (MPMD) are subcategories within the MIMD classification, differentiating whether the multiple instructions

are from the one or many programs. Within SPMD, the multiple processors execute instructions from the same program [80], MPMD involves the collaboration of multiple programs with differing behaviour.

Modern CPUs are primarily MIMD architectures, containing multiple complex processor cores within a shared-memory environment. They are well suited to task-level parallelism and coarse-grained data parallelism. Over time, multi-core processors are exhibiting higher degrees of parallelism, both in terms of core count but also the adoption of wide vector units, providing some SIMD instructions. Many-core processors, such as modern GPU are shared memory processors containing large numbers of relatively simple processors cores, often arranged in wide vector units. GPUs typically follow the SIMT paradigm, offering high levels of performance for fine-grained data-parallel applications. Although originally designed to accelerate the rendering of 2D and 3D computer graphics, they have been widely adopted for non-graphical tasks through General Purpose Computing on Graphics Processing Units (GPGPU) computing. This includes use within scientific computing, and large scale HPC systems, in part due to the energy efficiency offered by the GPU architecture. Heterogeneous architectures containing accelerator architectures account for 7 of the top 10 most performance supercomputers in the November 2020 top 500 list [81].

GPUs may be integrated within CPUs and share access to the same memory, however, high performance GPUs are generally implemented as co-processors with independent high-bandwidth memory, connected through an interconnect such as PCI-e [82] or NVLink [83].

Similar to GPGPU accelerators, Intel offered a range of many-core processors using the Many Integrated Core (MIC) architecture [84], [85]. These were originally co-processor accelerators much like GPU, but were later released in a more traditional CPU like form factor. Unlike GPGPU accelerators, the many-core MIC used the x86 instruction set, enabling the execution of traditional CPU software with fewer changes. Although the MIC processors offered similar theoretical performance to competing GPUs architectures for many applications [86], the architecture was discontinued due to limited uptake within the scientific computing and HPC communities.

2.2.2 Parallel Programming Models

There are many alternate approaches for developing software targeting parallel processing architectures, the choice of which depends upon both the parallel system being targeted and the trade-off between development time and application runtime. For shared-memory parallel architectures, parallelism can be achieved in several ways: the use of programming libraries (which may be included with a programming language) to parallelise commonly used algorithms; the use of user-provided compiler directives to instruct software compilers on sections of code to parallelise; or explicit parallelism through programming languages.

Programming languages such as C++, Fortran and Julia among many others [87]–[89] provide high-level access parallelism, through methods provided by the language or standard library. For instance, C++17 provides users with the ability to request parallel implementations of standard library algorithms through execution policies, specifying how the algorithm should be executed in parallel. For example, reductions over lists of numbers can be requested to be executed in serial or parallel [90], which may target CPUs or certain compilers provide GPU support [91]. Alternatively, third party libraries can provide high-level access to parallelism using similar structures, again capable of targeting multiple processor architectures [92]–[94].

Compiler directives are an alternate approach to provide access to parallelism to software developers, with OpenMP [95], [96] and OpenACC [97] being the main compiler directive standards. Directive-based approaches provide a simple technique to parallelise existing serial code. Compiler directives are embedded within application source code as a specialised type of comment. They instruct the compiler to execute the associated region of code in parallel, allowing the user to specify the degree of parallelisation which should be used, the scope of variables within the parallel region, and the application of parallel operations such as reductions of variables within the region. OpenMP and OpenACC can be used to target multi-core CPUs and GPUs (as of recent versions), supporting C, C++ and Fortran as programming languages. As they can target multiple architectures from multiple hardware manufactures, they offer some degree of *performance portability*, where the same source code can offer high performance across a range of hardware, however, there is a trade-off between portability and absolute performance, with explicit implementations able to extract the maximum performance from a processor.

The third approach for shared memory environments is to use programming languages which provide low level access to parallel execution. For applications targeting multi-core CPUs via multi-threading, programming languages such as C and C++ provide access to data structures and methods which allow new threads to be created, which can execute in parallel with other threads. These low-level approaches offer very fine-grained control to the programmer, at the cost of software complexity and typically increased development time and effort. Compilers can also provide ILP in optimised builds [76].

There are many options for low-level programming of many-core GPUs. As GPUs were originally used for 2D and 3D computer graphics, APIs such as Direct X [98], OpenGL [99] and Vulcan [100] provide compute shaders as a method of performing general purpose computing, however, these are typically only used within applications which already use the corresponding computer graphics API. Alternatively, explicit standards and languages exist which are either open, targeting GPUs from several competing manufactures, or closed targeting processors from a single manufacturer. These allow very fine-grained control for execution on the GPU, which may outperform directive based applications but require substantially greater development effort. This investment of effort may however be required for tasks which are not well suited to

the directive based approach. OpenCL [101] and SYCL [102] are cross-platform programming standards which target heterogeneous systems, offering performance portability for any hardware architecture with a supported driver. Unlike the directive based OpenMP and OpenAcc approach to performance portability, the OpenCL and SYCL provide more fine-grained control. OpenCL provides a C-style dialect for explicitly programming kernels which are executed concurrently in a data-parallel context. SYCL is a higher level language than OpenCL with a modern C++17 based dialect. It has many of the same aims as OpenCL, providing performance portability across a broad range of processing hardware.

The NVIDIA CUDA programming toolkit [103] in comparison is a closed, proprietary approach to GPGPU which only supports NVIDIA GPUs. The closed nature of the CUDA toolkit is generally regarded as a negative feature within scientific computing. However, it is one of the most widely used approaches for low-level GPGPU parallelisation. This is mainly due to a more comprehensive set of software libraries and more advanced debugging and profiling tools offered by the toolkit, as well as access to hardware specific features only provided by NVIDIA GPUs. Only targeting a single GPU architecture also has advantages compared to more performance portable approaches, with application developers only having to optimise for a single architecture, rather than either provide multiple optimised versions, or accept a loss of performance through a common code-base.

A number of research domains have considered these different approaches to many-core acceleration and compared the approaches to one another. It was generally found that the directive based approaches such as OpenMP and OpenACC required substantially less time and effort to provide a GPU implementation than more explicit approaches, with OpenCL requiring more effort than CUDA [104]. Benchmarks also showed that in some cases the directive based approaches could achieve the same (or at least very similar) levels of performance to the lower level approaches, although in some cases the directive based methods would only achieve 50% of the performance of a CUDA implementation [105] or even as low as 5% [106].

In distributed memory environments processors in one node cannot directly access the memory of processors in another processing node, regardless of whether they are CPUs, GPUs or a mixture. A method of communication between nodes is therefore required to orchestrate the work to be performed, and to transmit data between nodes. Message Passing Interface (MPI) is a portable, distributed communication standard for parallel computing [107], with several well-known implementations such as OpenMPI [108], MPICH [109] and MVAPICH [110]. The communication protocol provides methods for direct (point-to-point) and collective communication between processes, and may be used within a shared memory environment, or a distributed environment. Most implementations are provided as C/C++ and Fortran APIs, with bindings used by higher level languages. Within MPI applications, communication over low-bandwidth high-latency network connections can have a significant impact on application performance is

generally minimised where possible. As distributed systems with GPUs account for a significant number of the top HPC systems in the world. As GPUs are co-processors connected to a CPU-based machine, transferring data between GPUs in a distributed system can show have a considerable impact on software performance, as the CPUs and host memory will be involved in the transfer. To address this, technologies such as GPUDirect RDMA have been implemented [111]. GPUDirect RDMA allows devices connected to the PCI-e bus other than the CPU to access device memory. By including the network controller this can significantly reduce the latency of distributed memory transfers.

2.2.3 NVIDIA GPUs and the CUDA Programming Model

Of the GPGPU programming approaches, NVIDIA CUDA is used heavily within this thesis due to the availability of key libraries and tools, as well as the fine-grained control offered by explicit parallel programming methods compared to directive-based approaches. CUDA is the parallel computing platform and programming model for general purpose computing on NVIDIA GPUs [112]. It is implemented as extensions to C/C++ and Fortran, with the CUDA C++ being the main dialect. It was first introduced in November 2006, with support for the Tesla GPU micro-architecture. As previously discussed, GPUs are many-core accelerators, optimised for fine-grained data-parallelism and thread-parallelism following a SIMT model.

2.2.3.1 GPU Architecture

NVIDIA GPUs are designed as an array of multiple *Streaming Multiprocessors (SMs)* with up to 80GB of high-bandwidth memory (as of 2020). An SM is a multi-threaded processor, each containing many functional cores, multiple schedulers and multiple caches, which follow the SIMT execution model. Within an SM threads are managed, scheduled and executed in groups of 32 concurrent threads (for all current micro-architectures) known as a *warp*. Each thread within the warp has it's own registers and instruction counter, allowing independent execution and branching. Each warp executes one common instruction at a time, so maximum efficiency is achieved when all threads within a warp follow the same execution path. Different models of a GPU micro-architecture have different quantities of SM, different volumes of device memory and operate at different frequencies.

Each GPU contains gigabytes of high-bandwidth global memory, which can be accessed by all of the SMs within the GPU. There are also several much smaller memory cache's available to the SMs. Memory caches are relatively small regions of fast memory located close to processor core(s). When memory requests are made from lower-performance areas of memory, the values can be copied into a cache layer. If the contents of the requested memory address already resides within a cache (a cache hit) it is accessed with much lower latency than if the requested

memory was not already stored within the cache (a cache miss). Each SM has a read-only constant cache, which improves performance for accessing read-only data from a special area of global memory, and a unified data cache which combines an L1 data cache and an area of *shared memory*. Shared memory is fixed-size area of low-latency memory located near to each processor core, which is often used as a software managed cache [112]. Outside of the SMs there is a shared L2 cache which is used to cache local and global memory accesses. The exact sizes of these caches, and the ratio of L1 to shared memory are mircoarchitecture dependent.

Each SM contains many functional cores which operate on different data types. Recent architectures containing up to 128 FP32 cores, 32 FP64 cores, and 64 cores for integer operations. There are also specialised cores for mixed precision matrix arithmetic. GPU microarchitectures designed for primarily graphics purposes (such as those used in the GeForce line of consumer GPUs) contain many fewer FP64 cores than FP32, as 64-bit precision is not typically required for computer graphics, at a 1:32 or 1:64 ratio. Microarchitectures intended for processors used in scientific computing typically have a 1:2 ratio of FP32 to FP64 cores, with additional capability for reduced precision.

Most NVIDIA GPUs are dedicated co-processors, connected to a host CPU-based system via a PCI-e interconnect, or in the case of Power9 architecture a higher-bandwidth interconnected NVLink [79]. NVLink connections are also available for high-bandwidth connections between multiple GPUs within a multi-GPU system. These interconnects are of a relatively low-bandwidth compared to on-device or on-host memory, so care must be taken to minimise data transfer to and from a GPU.

2.2.3.2 CUDA Programming Model

The CUDA programming model is a *Scalable Programming Model*, which provides abstractions for a hierarchy of thread groups, shared memories, and synchronisation [112]. In this programming model, code to be executed on the GPU is specified as a *kernel* - a function to be executed on the GPU N times in parallel using N threads, following a thread hierarchy. The thread hierarchy is made up of *threads*, *blocks* and *grids*. Individual threads are grouped into a one, two or three dimensional block of threads. Each thread block is of a limited size, for most current micro-architectures this is 1024 threads per block, shared across all dimensions. Many thread blocks are arranged in one, two or three dimensions to form the grid. Each block of threads within a grid must have the same dimensions. This block-level distribution of work onto the SMs of the GPU allows the same applications to be executed on GPUs with varying numbers of SMs. Figure 2.4 illustrates the CUDA thread hierarchy for an example grid of 6 thread-blocks arranged as 2 rows of 3 columns, where each 2D block contains 8 threads arranged as 2 rows of 4 columns.

When a kernel is launched, thread blocks are dispatched to the SMs of the GPU with

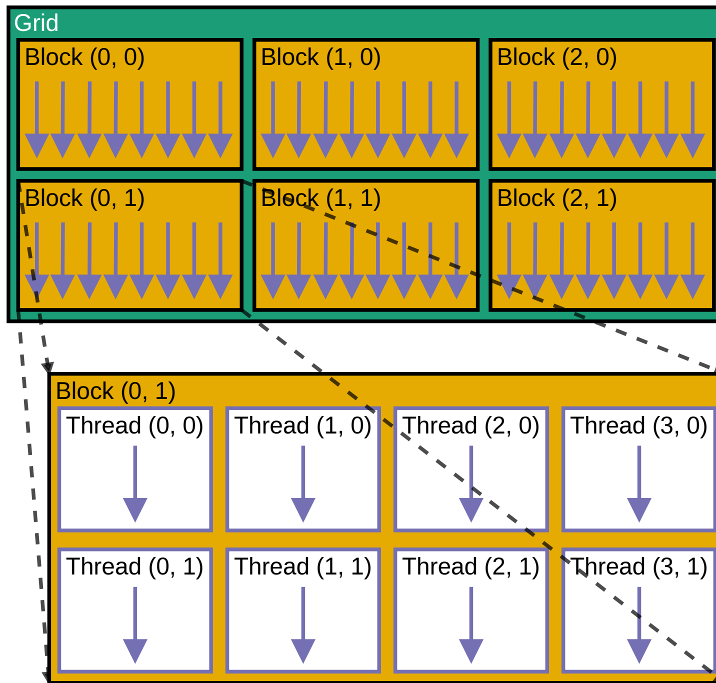


Figure 2.4: An illustration of the CUDA thread hierarchy, for a 2D grid of 6 blocks, where each 2D block contains 8 threads. The arrangement of threads within a 2D block is shown for block `(0, 1)`.

remaining capacity. Once a block is assigned to an SM, the threads are considered *resident*, with the execution context for the block occupying space within the register file, program counters and shared memory area of the SM. They remain resident until each thread within the block has completed. Each SM can support multiple resident thread blocks which come from one or more grids, subject to the resource requirements of the kernel and the specific micro-architecture of the GPU. The order in which blocks are assigned and executed is not guaranteed, and may occur concurrently or in serial. Each block should therefore be independent from one another in most cases. The thread-hierarchy also allows the same kernel to be executed using different models of GPU which may have differing numbers of SMs. For instance, a grid of 6 blocks of threads can be executed by a 2 SM GPU using 3 waves of execution, or in 2 waves using a 4 SM GPU. This is illustrated by Figure 2.5.

Resident threads occupy limited resources of the SM, such as the register file and shared memory. As these resources are limited, the number of threads which can reside on each SM at any given time is also limited. There are many factors which impact the number of resident threads for an SM. For a given kernel, the number of registers used per thread impose a potential limit, as does the shared memory use. There are also limits on the number of blocks, the number of threads or the number of warps per multiproccessor. These factors when combined impact the *theoretical occupancy* of the multiprocessor which can be achieved by a given kernel. In general, higher occupancy leads to improved performance as there are more opportunities for latency to be hidden by the warp schedulers although this is not always the case. The CUDA

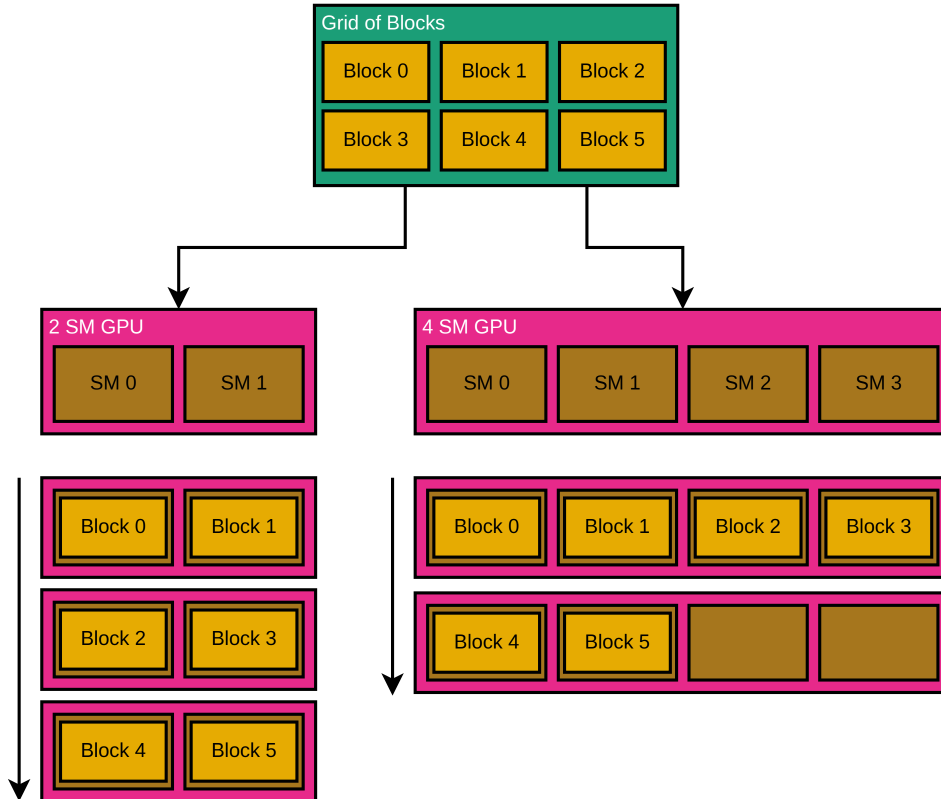


Figure 2.5: An illustration showing how the CUDA thread hierarchy allows a kernel using a grid of 6 thread-blocks to be executed on GPUs with 2 or 4 SMs.

Occupancy API [112] provides methods to find block sizes for a kernel which will lead to the highest theoretical occupancy, and usually good levels of performance.

On top of occupancy, the utilisation of the GPU is important to consider when launching a kernel. In the case of an application which only launches a single kernel, the total number of threads and the shape of the grid-block hierarchy needs to result in at least one block of threads being assigned to each multiprocessor to ensure that the full GPU can be occupied. For example, an NVIDIA V100 GPU contains 80 SMs, each of which can have 2048 resident threads, resulting in $163,840$ threads being required to fully occupy the GPU (if there are no resource constraints). As the number of multiprocessors within GPUs is variable, it is common to oversubscribe GPUs by launching more threads than can be resident within the SMs of the device. Although this will result in serialisation of blocks, as new models of GPU are released which can support larger numbers of threads the performance of an application may improve without having to make any changes to the parallel application.

When an SM is ready to issue instructions each warp scheduler within an SM selects a warp which is ready to execute the next instruction. The switch between execution contexts of resident warps has no cost, as the execution context is maintained for the lifetime of the block of threads. This can be leveraged by the application runtime to hide instruction or memory latencies. As CUDA follows a multi-threaded SIMT programming model, threads within a block

or warp may diverge from one another, but as each instruction is executed for all threads within the warp this can reduce performance. The threads within a warp which are participating in an instruction are considered the *active* threads. Threads may be inactive due to branching such as conditional statements, earlier termination due to return statements, or the final warp(s) of the block may contain a number of inactive threads when the block size is not a multiple of the warp size. The impact of inactive threads on performance can be significant, by reducing the efficiency of the SM. Figure 2.6 illustrates the impact of branching on SIMT programming model. The Volta GPU architecture introduced a revised SIMT model with *independent thread scheduling* [79], which is enabled through the use of the appropriate compiler flags. This changes how the SM groups active threads from the same warp together for execution, when they may have diverged from one another, offering sub-warp divergence and reconvergence, which enables some finer-grained algorithms to be implemented which were difficult to accomplish with the branching model.

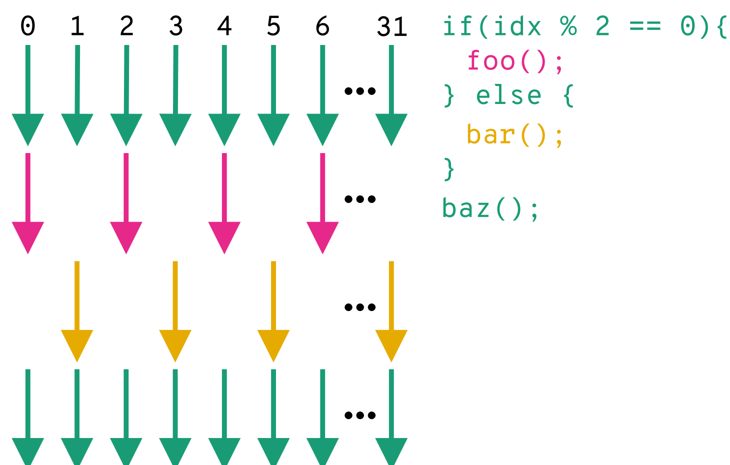


Figure 2.6: An illustration of divergence occurring within an SM following the SIMT GPU architecture. Threads with odd indices perform separate operations to those with even indices, and therefore must be executed independently, prior to re-converging for a warp-wide operation. GPU Architectures which support independent thread scheduling will not necessarily reconverge without explicit synchronisation.

Within GPU kernels, the use of the memory can have significant impact on application performance. Global memory is the large volume of device memory accessible to all threads. It is the highest volume memory area available on the GPU, but accesses to it are high latency. Accesses to global memory from the threads within a warp should be coalesced to maximise throughput. Memory accesses are considered to be coalesced when consecutive threads access consecutive elements of memory. If threads do not access consecutive elements of memory, the accesses are considered non-coalesced or scattered. For example, if four consecutive threads 0, 1, 2 & 3 read memory elements at indices 16, 17, 18 & 19 this would be a coalesced coalesced read. However, if instead these threads accessed indices 4, 12, 7 & 19 this would be a non-coalesced or scattered read. Coalesced memory accesses offer improved performance compared to scattered accesses, by reducing the number of transactions required to complete the necessary memory

operations. Global memory transactions may be 32, 64 or 128 bytes wide, and may be cached through the L2 and L1 caches depending on the GPU architecture.

To minimise the use of high-latency global memory, several memory access methods are provided. Most current devices (with compute capability 3.5 and above) can access data which does not change over the lifetime of a kernel through the *Read-only data cache*. The compiler may be able to automatically detect that certain accesses should be performed through this cache, but otherwise it may be hinted via memory qualifiers, or directly instructed through the `__ldg` intrinsic method. In most architectures, this resides within the unified data cache along with the L1 cache. Repeated accesses to data stored through this cache can offer reduced latency and higher bandwidth than through global memory alone.

Memory latency can also be reduced through the use of a relatively small area of on-chip, per SM, shared memory, which can be accessed by threads within a block. It is located within the unified data/L1 cache on most current architectures, offering reduced latency and higher bandwidth than for global memory accesses. The shared memory is divided into equally sized modules called banks, which may be accessed simultaneously, however, if multiple concurrent requests are made to the same bank throughput is reduced. As shared memory is accessible to all threads of the warp, it can play an important role in collaborative work.

The constant memory space is an area of global memory which can be used to store values which do not change within a kernel, and values must be set from the host prior to kernel launch. SMs have a read-only constant cache which improves throughput when accessing constant values, if a cache hit occurs. This dedicated cache is most suitable for broadcast operations, where the threads within a warp all access the same memory address concurrently. Only 64KB of constant memory is available.

Registers provide the lowest-latency storage available to threads within an kernel, but each thread can only be allocated up to 255 registers, although typically fewer are allocated to support larger block sizes. If a thread requires more registers than available, they will be *spilled* to per-thread *local memory*, which resides in the *global memory* of the device with much higher latency and reduced bandwidth. As such, register spilling should be avoided where possible.

In addition to using the most appropriate part of the memory hierarchy for the task at hand, the data structures stored within memory may need to be revised to maximise performance. To achieve maximum bandwidth from global memory, memory accesses should be coalesced, with neighbouring threads accessing neighbouring elements of memory. This can be achieve by using Structure of Arrays (SoA) data structures rather than an Array of Structures (AoS). In some cases it may be worth considering interleaving multiple elements of data within the SoA, to form a Array of Structures of Arrays (AoSoA) [113].

GPUs are often connected to a host CPU with its own memory, the CUDA programming model provides mechanisms for transferring data to and from the device. This can be performed

explicitly through synchronous or asynchronous operations, or since CUDA 6.0 through *unified memory*. Unified memory is a managed memory space, providing memory coherency between the host and device memory, with a common address space. This greatly simplifies data management compared to the explicit memory management approach, but may result in reduced application performance. More recent CUDA architectures (Pascal and above) have improved unified memory features, including over-subscription of memory, or on-demand page migration but these features are only supported with some operating systems.

As with any parallel programming model, synchronisation is important. Within a function executing on the GPU, the threads within a warp, or threads within a block may be explicitly synchronised. For example, this may be important when threads within a block are writing to and reading from shared memory, in which case synchronisation can be used to ensure that all threads have finished writing prior to any read operations being performed. CUDA also provides *atomic functions* to support transactional operations on elements of memory without interference from other threads.

From CUDA 9 and Pascal architecture GPUs, it is possible synchronise across the entire grid of threads or even across multiple devices through the cooperative groups API [112], however, grid-wide and multi-device synchronisation are much more costly than block-wide synchronisation, with additional limitations.

Outside of kernels, the CUDA programming model provides *streams* as a method of performing asynchronous operations. Multiple operations such as memory transfers or kernels may be assigned to a stream. Operations within a given stream will execute in-order, but will be asynchronous with respect to other streams or the host CPU thread, with explicit synchronisation of streams required. An alternate directed acyclic graph based approach has been provided in recent CUDA releases.

2.2.4 Summary

This section has described the many available approaches to the parallelisation of software, including a more in-depth view of many-core GPUs which can offer performance advantages over general purpose CPUs. As shown previously Section 2.1, road network simulations can be computationally demanding, with long software run-times imposing limits on use. The next two sections, Sections 2.3 and 2.4 provide a more detailed review into microscopic and macroscopic road network simulation, including the application of parallelisation to improve performance

2.3 Microscopic Road Network Simulation

Microscopic models of transport networks are fine-grained models following a bottom-up approach of modelling each individual within the simulation. Agent Based Modeling (ABM) is one

possible approach for microscopic modeling, with the terms often used interchangeably within the context of road network simulations [114], [115]. Microscopic road network simulations are traditionally only used for small-scale simulations, due to the computational cost. However, they are well suited for certain tasks at this scale, such as the optimisation of traffic signal timing [116], modelling the effects of driver behaviour on vehicle emissions [5], [117] or modelling new modes of transport such as CAVs [118], [119]. Such modelling could not be achieved using top down approaches.

2.3.1 Architecture of a Microscopic Road Network Model

Microscopic road network simulations typically model the transport network at the individual level. When using ABM, agents are used to represent individual vehicles, pedestrians, cyclists or other modes of transport. Agents can also be used to model complex pieces of transport network infrastructure such as traffic lights or various sensors found within transport networks. The transport network itself is often represented as a graph.

For road networks, the individual vehicles within the network are modelled, each with individual properties and behaviours. The individual vehicles can interact with one another, and with the environment to achieve their goals, such as safely reaching their destination. The environment can be made up of individual agents modelling complex parts of road network infrastructure, such as sensors or traffic lights, whilst the road network itself is often represented as a directed graph, encoding information about the properties of the road such as the physical location, gradient or speed limit.

Road network microsimulation packages such as SUMO [120], PTV VISSIM [121] and Aimsun [122] provide many tools which are used to prepare, execute and analyse transport networks. They are capable of modelling the wide range of road network infrastructure which is in use in different locations across the globe, often providing multiple alternate models for aspects of the transport network simulation. The modelling and simulation components of these application suites typically take the transport network, model parameters and demand information as simulation inputs. Tools may be provided to prepare these inputs for corresponding simulation engine, or to import them from formats used in other simulation tools. Multiple simulations of the transport network are then executed using the simulator, the outputs of which can be fed into analysis tools to aggregate statistical properties of the network to present to decision makers, or they may be visualised in 2D or 3D computer graphics applications.

Microsimulation packages are often support multiple modes of transport, such as pedestrians, bicycles, road vehicles and public transport. Each individual or vehicle within the microsimulation will have local properties such as the physical size of a car or bicycle, the maximum acceleration of the vehicle. They will each have their own model parameter values, subject to the specific models they implement, such as a vehicles perception of a speed limit. These values

are often sampled from parameter distributions, which are based on real-world distributions of those properties or are generated through calibration procedures.

The individual elements of dynamic road network infrastructure are also modelled. For instance, traffic signals may be modelled individually, or many signals may follow the same timing cycles as one another, subject to the environment being modelled. Sensors such as induction loops may also be modelled, which interact with other agents within the simulation. The modelling of individual junction types and the operational optimisation of these is an active area of transportation research [123]–[125].

For road network simulations, the network of interconnected roads in which individual vehicles exist must also be represented within the model. Road networks are typically represented using directed weighted graphs, as discussed in Section 2.1.3. These graphs must be prepared so that they represent the real world network in a sufficient level of detail for vehicles within the simulation to interact with in a realistic fashion. Vehicle demand information is often associated with the underlying transport network. This may be presented in the form of Origin Destination (OD) demand data, dictating how many vehicles of a given type will travel from one point in the network to another with the routes taken by individual vehicles being based on shortest paths through the network, or pre-determined routes input by the modeller. Alternatively, the network may encode route selection through properties such as turning proportions, where random number generation is used to dictate which vehicles will make a given turn from one road to another.

2.3.2 Road Network Agent Based Models and Properties

When considering individual vehicles or components of network infrastructure within an ABM, specific behaviours can be modelled at varying levels of detail. For instance the acceleration of a vehicle could be modelled as a fixed value, or a more complex process based on the mechanical properties of the vehicle in question. As such, there are many individual models which can be incorporated into an ABM of a road network, but some of these behaviours are more prevalent than others in research literature and commercial tools used within industry.

Individual vehicles within a road network simulation have their own properties such as the physical dimensions of the vehicle, properties such as the acceleration and deceleration characteristics of the vehicle or the accuracy of perception of the driver of the vehicle. Different models will require different properties be available. The following subsections describe research into microsimulation behaviours for transport simulation.

2.3.2.1 Car Following and Stopping

On single lane roads, or within an individual lane on sections of road with multiple lanes vehicle drivers wish to drive at their desired speed without colliding with the vehicle ahead [126]. The car following theory is that a driver reacts to the behaviour of the car(s) in front. Early models were mainly concerned with the trailing car's acceleration being proportional to the relative speeds of each car at an earlier time, or a 'history' of relative speeds [127]. Later models took into account the reaction time of the vehicle driver [128], as there is a delay between an individual receiving stimuli and performing the correct action. Performance limits of both the driver and vehicle also effect car following behaviour, safety conscious drivers will ensure there is a gap large enough in which to stop between them and the vehicle in front (limited by an acceptable level of braking). Vehicles also have limits of acceleration performance, and so a trailing car will accelerate with no more than the vehicles maximum acceleration until it nears the desired speed (the speed of the vehicle in front) at which point the rate of acceleration will decrease to zero [128].

More complex models, based on the idea of car following have since been developed, such as the *Intelligent Driver Model* proposed by Treiber et al in 2000 [129]. At it's core this model bases the acceleration of the trailing vehicle on the ratio between the "desired minimum gap" and the actual gap to the vehicle in front, with special cases for scenarios such as when traffic is in equilibrium where drivers will keep a gap to the vehicle in front which is dependant on velocity.

The literature contains a wide range of car following behavioural models which are implemented within road network simulations used in literature and in industry [130]–[135], with new additions and extensions continuously being investigated. New models may include additional vehicle properties, such as the illumination of vehicle brake lights from the lead vehicle during deceleration [136]. Or they may attempt to capture more advanced behaviours demonstrated by road network users, such as the asymmetry shown in vehicle following between acceleration and deceleration [137]. New modes of transport such as CAVs must be accounted for, which will exhibit different behaviours to non-autonomous vehicles, with several new car following models being proposed for the new mode of transport [118], [119]. The car following model may also be the source of vehicle deceleration and stopping at road network infrastructure such as as stop signs or traffic lights, due to the similarity between the deceleration behaviour and that of approaching a stationary vehicle.

2.3.2.2 Lane Changing

Road users often have to change lane, in order to head in their desired direction or overtake slower or stationary vehicles [138]. Lane changing decisions can be more complex than some

behaviours such as car following, a driver may wish to be in a given lane to make a desired turn, but be forced to change to a less favorable lane in order to avoid an accident [126]. Gipps proposed that the decision to change lanes depends on the answer to several questions a driver must ask: is the lane change possible, is it necessary and is it desirable.

Gipps also suggested three patterns which drivers fit into based on the distance to the desired turn. While the turn is remote it does not effect lane changing behaviour, which is focussed on speed. At distances considered "middle distances" by the driver speed gains by changing lane in the wrong direction will be ignored, and at short distances to the turn the driver will remain in lanes appropriate for the turn where possible [126]. Kesting et al proposed the "politeness factor" - as a way of modelling driver aggressiveness when changing lanes within the MOBIL lane changing model [139]. This factor enabled their model to allow for polite drivers who do not wish to obstruct others by changing lanes, as well as aggressive drivers being able to induce lane changing of slower drivers. As with car following models, there are many lane-changing models defined in the literature. These range in complexity, from intentionally high-level approaches to reduce the frequency of lane changes occurring [140], models which aim to more accurately represent behaviour in specific cases, such as overtaking on two-lane rural roads [138] or to limit the lanes used based on the intent of the driver [141]. Emerging computational techniques such as deep learning have also been applied to transport modelling in order to improve the quality of models, for example to improve the accuracy of lane changing behavioural models [142].

2.3.2.3 Junction Modelling

Road networks can contain many different types of junction which must be accurately modelled for simulations to produce meaningful results. The behaviour at junctions within a road network are often influenced by the local rules, which can lead to many alternate implementations to support modelling of different countries. For instance, in some locations there there are no acceptable reasons to enter a junction while a red light is shown, while in other jurisdictions vehicles are allowed to make right hand turns during red phases while following other give-way rules.

A common approach, which is also a factor in many microscopic road network behaviours, is that of *gap acceptance modelling* [143], [144], while following the appropriate right of way rules for the scenario being modelled. From the perspective of a vehicle at turning from a minor road onto a major road, who has to give way to other vehicles, the gap is the time interval between two successive cars in the stream of traffic. The vehicle will wait for a sufficiently large gap between vehicles for it to safely make it's movement, crossing or joining the flow of traffic. The size of the required gap depends on several factors, such as the speed of vehicles, the rules of the intersection in question, the maneuver being executed, road network geometry and the properties of the vehicle performing the maneuver [145].

Gap acceptance models often assume that priority is absolute, with minor flows always conceding to major flows at unsignalised junctions. Instead, some models propose that the priority of a major flow should be limited allowing minor flows a higher rate of success in finding acceptable gaps, improving the ability for gap acceptance models to accurately reflect some real world conditions [146], [147]. As there are many forms of junction, there are many variations of gap acceptance models which aim to improve accuracy for differing scenarios, such as motorway merges [148], [149], roundabouts [150] or three-legged intersections [151].

2.3.2.4 Dynamic Road Network Infrastructure

Modern transport networks contain many pieces of dynamic transport network infrastructure. Traffic signals may follow fixed schedules, may be centrally controlled or may respond to inputs from sensors embedded within the road network [152]. Intelligent transport systems such as smart motorways in the UK may have dynamic speed limits to influence the flow of traffic, or to close lanes as a response to road traffic collisions [153]. For the impact of changes in control flow for these systems to be evaluated, they must be modelled and simulated accordingly, with road user behavioural models responding appropriately.

2.3.3 Microscopic Road Network Simulation Software

Some widely used CPU-based micro-simulators such as SUMO [120], [154] only provide sequential implementations. Current state of the art microscopic simulation tools such as Aimsun [143], MATsim [155], PARAMICS [156] and PTV Vissim [121] leverage multi-threading on multi-core CPUs to provide improved simulator performance. SUMO [120], [154] is an open source road network simulation package, providing microscopic and mesoscopic road networks simulations with a broad range of available models to choose from. The simulations are executed sequentially using a single CPU processor core resulting in very long application runtimes for larger scale models, however, it is widely used within literature due to the open source licence and flexibility of modelling approaches. Aimsun [143] is a commercial road network simulation suite which provides microscopic and mesoscopic simulations. Task parallel and coarsely-grained data-parallel algorithms are applied to multi-core CPUs to improve performance of the computationally demanding fine-grained simulations. PARAMICS [156] is a parallel microscopic road network simulation package originally developed to evaluate the application of HPC techniques on microscopic road network simulation. Several parallel implementations have been released, including a data-parallel SIMD implementation, and an MPI based implementation targeting distributed systems. VISSIM also leverages multi-core CPU parallelism to improve simulation performance and reduce application runtimes. Although PARAMICS investigated a SIMD implementation in the 1990s, none of these commercial or open source tools leverage many-core

processors to increase runtime performance.

GPUs have been used previously in microscopic transport network simulation. They have been implemented as both cellular automata [23], [26] showing some performance improvements over CPU based implementations. Agent-based models have also been described within the literature, [24], [26] offering performance improvements of 11x for a car following model based implementation. The suitability of GPU-offloading as an approach to achieve performance portable road network microsimulations were investigated by Xioa et. al [25]. The implementation suggests only performing part of the simulation on the GPU does not offer significant performance advantages over just using directive based techniques to implement multi-threading on the CPU, however, an more complete OpenCL implementation did achieve a 28x performance improvement. Yedavalli et. al [157] recently published results for region-scale simulations performing much faster than real time. This was achieved by discretizing the road network into short chunks of a fixed size to provide an efficient mechanism to map the spatial distribution of vehicles within the road network. Additionally other tasks associated with microscopic simulation have been accelerated using GPUs, such as the generation of demand data using activity plan generation [158], [159], traffic signal optimisation [160] and dynamic route assignment [161]. Recently, outside of microscopic simulation, GPUs have been utilised within mesoscopic road network simulation tools and performance improvements have been demonstrated [162]–[165], further highlighting the demand for reduced software run-times within the transport modelling sector.

2.3.4 Agent Based Modelling Frameworks

Agent-based frameworks and toolkits can aid the development of ABM for researchers and modelling practitioners who are not specialist software developers. They can provide functionality to handle the data structures and complex algorithms which can be used to provide performance in order to offset the high computational complexity of fine-grained simulations. In effect they provide a separation of model and implementation. Frameworks may offer alternate perspectives to one another on how agent based models should be structured, either to improve usability of the modelling approach, or to offer greater levels of simulation performance. The frameworks may also be well-tested, potentially resulting in higher quality simulations with fewer software bugs.

There are many agent based modelling frameworks which could be used to implement microscopic road network simulations such as MASON [166], NetLogo [167], Repast [20], [168], FLAME [169], [170] etc, often providing multiple implementations which target alternate processing architectures. The majority of these ABM frameworks target CPUs, with a mixture of sequential, multi-threaded and distributed approaches. Most of the MASON [171] and D-MASON [166], [172] are frameworks implemented in the Java programming language, multi-core

CPUs and distributed systems. The distributed implementation leverages MPI to distribute the agents over multiple processor nodes, increasing the potential scale of simulations and reducing application runtime. Recently a cloud-based variant of D-Mason has been introduced, with improved algorithms for distributing the agent-based model across many CPU-based processing nodes [173]. Repast [168], [174] provides a rich interactive agent-based modelling platform, while Repast HPC [20], [175] provides an MPI-based distributed implementation targeting high performance when executed on large CPU-based high performance computing clusters. Flexible Large-scale Agent Modelling Environment (FLAME) [170] and FLAME-II [176] are CPU-based ABM frameworks which use a formal model, called a communicating X-machine, to model complex systems. X-machines are extended Finite State Machines which include memory, inputs and outputs [177], [178].

Modelers specify a set of states which agents can exist in, and functions which transition agents from one state to another, forming the X-machine. FLAME and FLAME II were designed with parallelism in mind. The original version of FLAME uses distributed computing to support larger models and improve performance, distributing the agents within the simulation across the processor nodes. Agents communicate through message boards, which allow parallel communication following the structure of the X-machine. FLAME-II sought to address some of the limitations of FLAME, by increasing the degree of parallelism exposed through the state machine, and support for multi-threading on modern CPUs.

There are now several GPU-based general purpose ABM frameworks which enable fine-grained data-parallel ABMs without the need for the specialist programming knowledge required to implement a GPU implementation including FLAME GPU [179] and MCMAS [180]. Flexible Large-scale Agent Modelling Environment for Graphics Processing Unit (FLAME GPU) is an extension to the FLAME family of general purpose ABM frameworks, providing high levels of performance through a CUDA implementation targeting NVIDIA GPUs, and a state-based representation of agent dynamics which is well-suited to the SIMT architecture. The complexities of GPU programming are abstracted away from the modeller, removing the need for specialist knowledge to access the high levels of performance which can be provided by many-core architectures [178], [179], [181]. FLAME GPU has successfully been used in several modelling domains such as pedestrian crowd simulation [182] and cellular simulations [178], [183], [184] and investigations into emergent population dynamics [185]. A more in depth look into FLAME GPU is provided by Chapter 3. MCMAS is a toolkit for Multi-Agent Systems (MAS) on many-core processor architectures, leveraging OpenCL to execute portions of the multi-agent system on GPUs or CPUs in parallel, improving simulation performance.

In addition to these simulation implementations, there have been collaborative attempts to define a Domain Specific Language for parallel and distributed Agent Based Simulations (ABSs), which would improve the portability of a model implementation by allowing a single

model description to be implemented within any of the supported simulation frameworks [186].

2.3.5 Summary

This section has described the structure of microscopic road network simulations, and covered some of the key behavioural models which can be implemented within a microscopic simulation. A summary of road network simulation packages used within industry and literature is provided, along with a summary of popular ABM frameworks which could potentially be used to implement a microscopic road network simulation and how they leverage parallelism to improve performance. This will be further investigated within Chapter 3. Next, in section 2.4 a detailed view into the higher-resolution macroscopic road network modelling approach is provided, including shortest path algorithms which can have a significant impact on the performance of a macroscopic model.

2.4 Combined Macroscopic Road Network Assignment and Simulation

Macroscopic road network models and simulations are coarse models which follow a top-down equation-based approach. Rather than simulate individuals or groups of vehicles, the equation based models are applied to the sections of road which form the transport network, often based on the flow of traffic along each section [187]. Vehicle demand is assigned to the transport network, which is then simulated on a per-road-section basis with relatively large time-steps. Macroscopic models are traditionally the dominant approach used within road network simulation when evaluating changes to road network infrastructure. This is due to the relatively low computational cost compared to finer-grained modelling approaches [6]. However, macroscopic assignment and simulation models can still exhibit prohibitively long application runtimes for large scale models.

2.4.1 Architecture of a Macroscopic Assignment and Simulation Model

Macroscopic road network modelling is often split into two separate phases. Assignment is the process of determining how road network demand will distribute over the road network [188], while the simulation phase models the flow of assigned transport through the network, to evaluate the impact of the assigned demand on the transport network, such as queue lengths or the the delays at junctions [189]. These phases may be implemented as separate sequential processes, or as combined assignment-simulation models which iteratively perform assignment and simulation as a part of a convergent process. Convergence is based on the principle of Wardrop's equilibrium, which states that traffic will settle into an state of equilibrium where

no driver can reduce their journey time by choosing an alternate route [190]. Equilibrium can be reached following the iterative Frank-Wolfe optimisation algorithm, which minimises an objective function through subsequent linear approximations until the convergence criteria are met [191]. Several modifications of the Frank-Wolfe have been proposed in the literature to find more precise solutions [192]–[195]. Alternatively, other convergence algorithms have been proposed [196], [197].

Applications following the combined assignment-simulation macroscopic modelling approach approach can typically be decomposed into multiple phases: (i) Input; (ii) Pre-Processing; (iii) Assignment-Simulation Loop; (iv) Post-Processing; (v) Output. The following subsections explore each of these.

2.4.1.1 Input

The input phase of the application is the loading and validation of input data, such as the transport network, vehicle demand information and model parameters into the data structures used by the application. As discussed in Section 2.1.3, road networks are usually represented by directed weighted graphs, which are usually very sparse, with high diameters due to the low average degree of vertices, caused by the physical structure of road networks. It is common within macroscopic modelling approaches for zones or centroids to be used as virtual locations within the transport network, as locations for the modelled vehicle demand to enter (the origins) and exit (the destinations) the network. Zones can form a subset of the vertices within the graph network representation. The demand information is often encoded as a collection of OD matrices, specifying the number of vehicles which will traverse from an origin vertex to a destination vertex. The different modes of transport which make use of road networks will make different numbers of trips between origins and destinations to one another, which can be represented by a different matrix, or a by performing mathematical operations over existing demand information. The weights of edges within the road network graph may also need to be different for alternate vehicle classes, to account for rules imposed on the road network, or physical properties of the class of vehicle. For example, Figure 2.7 illustrates a small section of a road network containing 3 sections of road and a single junction. Two of the sections of road consist of a regular lane usable by any class of vehicle, and a bus lane which is restricted to only a subset of all vehicle classes. Graphs representing this section of road network for two different classes of vehicle ("Car" and "Bus") are shown. Edge weights on these graphs are used to represent the travel time down the section of road. In this example the travel time for the "Bus" class of vehicles from vertex `A` to vertex `B` is shown as 10 units of travel time, compared to 20 units of time for the "Car" class of vehicles, due to the presence of fewer vehicles using the restricted access bus lane. The input phase will also include validation of the input data, to ensure that it is appropriate for use. For instance, negative edge weights may be discarded

from the graph if they are used to represent a travel time along a section of road.

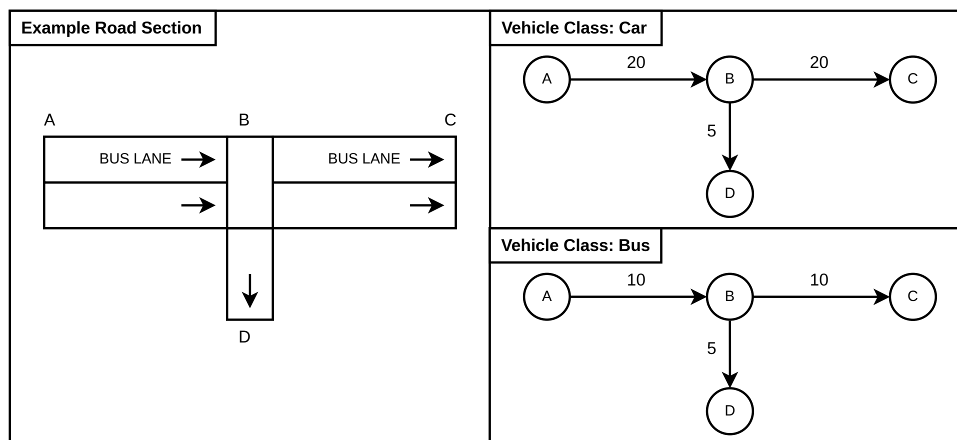


Figure 2.7: An illustration of a section of road network containing 3 sections of one-way road, two of which have vehicle class restrictions. The road network graphs representing these sections of road are shown for two vehicle classes, with edge weights representing travel time along the section. Due to the presence of the bus lanes, the road network graphs have differing values for each vehicle class.

2.4.1.2 Network Pre-Processing

After the simulation inputs have been loaded, a pre-processing phase may perform a range of tasks prior to the main assignment-simulation loop. This may involve checking for errors in input files; the mutation of input data; the re-arrangement of data structures to ensure that it is suitable for the techniques used within the assignment simulation loop; or make modifications which may improve the performance of the application without altering the simulation output.

For instance, the characteristic properties of road network graphs can restrict the performance of graph processing algorithms, which may favour denser graphs or graphs with lower diameters compared to the original transport network. A series of processes can be applied to generate a denser representation of a network graph for road network assignment. Edges or vertices of the graph which will not play an important role within assignment may be removed [198], for instance, dead-ends which do not terminate in an origin or destination zone may not be required. A denser representation of a graph can be found by replacing sequences of multiple edges with a single edge, while maintaining the cumulative weight of the replaced edges, essentially adding shortcuts to the network [198]–[201]. The new graph is known as a *contraction hierarchy* or *spider web network*. This process can be applied iteratively, forming denser and denser representations of the graph, however, the generation of the contraction hierarchy can be a time consuming process [201] and there are additional costs with switching between representations. There is a trade-off to be made between the cost of the generation process, and the performance improvements it provides to graph algorithms such as shortest path calculations.

2.4.1.3 Assignment-Simulation Loop

Provided the pre-processed simulation inputs, the iterative process of assigning vehicle demand onto the transport network and performing equation-based simulation of that transport network is applied repetitively until a converged, steady state can be found. This iterative process accounts for the majority of the runtime for macroscopic assignment and simulation models.

The assignment phase calculates the expected flow of traffic through the transport network based on the input demand data, and the state of the transport network from the previous iteration. For each trip between origin-destination pairs in the OD matrix, the route through the network with the lowest cumulative cost is found (i.e. the shortest paths). The shortest paths are then used to map the vehicle demand of the trip matrix onto the edges of the road network, to be used as input into the simulation phase of the iteration. The mapping of the demand onto the current shortest path can be applied in an all-or-nothing approach, where all vehicles travelling from point A to point B will take the same route, or fractional approaches can be taken, which assign a fraction of the demand to the current shortest route, updating edge-weights and then re-calculating the shortest paths to assign the next fraction of the each trip. The paths may also be stored for more detailed analyses. This process must be repeated for each class of vehicle being modelled. In many traffic assignment models, the shortest paths calculations can account for a significant portion of application runtime [201]. Section 2.4.2 discusses algorithms used for shortest path calculations, and how they related to both road network assignment and parallel processing.

The simulation phase of each assignment-simulation loop within macroscopic models often follow equation-based modelling approaches, although finer-grained elements may be incorporated to ensure certain vehicle behaviours can be captured, such as the blocking of vehicles at junctions [189]. The simulation phase takes the model parameters and the transport network with assigned vehicle demand as inputs, and calculates the resulting delays over sections of road, queue lengths and computed flows over the network. Within the simulation phase of the macroscopic model, mathematical equations based on real-world observations are used to simulate the flow of traffic through the transport network. For instance, the flow of traffic downstream of a signalised junction is observed to be cyclical, due to the periodic nature of junction signals. This can be approximated mathematically [152], [202], [203]. Alternatively, finite differential equations can be used to model the observed behaviours, such as the relationship between vehicle density and traffic speed [42], [204], [205].

The simulation phase of each assignment-simulation loop may itself involve iterative processes. For instance, vehicles will block one another from progressing through the junction if conflicting turns are being made. Rather than using finer-grained modelling of individual vehicles at the junction to capture this behaviour, convergent iterative processes can be used

instead. Delays, or the absence of delays, at one junction will impact the neighbouring sections of roads and junction in both upstream and downstream directions. The same time step can be simulated multiple times, using the junction related delays from the prior iteration. When subsequent iterations do not result in significant changes, then convergence has been achieved [202].

Although less computationally demanding than finer-grained simulation approaches the simulation phase of macroscopic road network models can still consume a significant portion of application runtime, especially when models approximate finer-grained models through iterative processes.

2.4.1.4 Post-Processing and Output

The data from the final iteration of the assignment-simulation process is then processed to generate the output data of the simulation. These post-processing steps may involve the reversal of any pre-processing which were performed, or additional processing using the state of the network may be carried out for more detailed analysis which is not required during the assignment-simulation phase, such as Select Link Analysis [206]. The final output data is often output per section of road, but also aggregate values of key measures may be provided.

2.4.2 Shortest Path Algorithms for Road Network Assignment

The shortest path between a source vertex and a destination vertex in a weighted graph is the sequence of vertices (or edges) between the source and destination with the lowest cumulative cost. For a directed weighted graph G containing the set of vertices V and set of directed one-way edges E connecting two vertices with real-valued weight function $w : E \rightarrow \mathbb{R}$, a path P of length $n-1$ from vertex v_i to v_n is defined as the sequence of vertices $P = (v_1, v_2, ..., v_n) \in V \times V \times ... \times V$ where there is a directed edge $e_{ij} = (v_i, v_j)$ between successive vertices in the path [207]. The shortest path between source vertex v and destination vertex v' can be defined as the path $P = (v_1, v_2, ..., v_n)$ where $v_1 = v$ and $v_n = v'$ which minimises the sum $\sum_{i=1}^{n-1} w(e_{i,i+1})$ over all possible n. For undirected graphs, edge weights are uniformly treated as having a value of 1.

Shortest path algorithms can be classified as either (i) Single Source Shortest Path (SSSP) algorithms, or as (ii) All Pairs Shortest Path (APSP) algorithms. The calculation of these paths can account for a large portion of macroscopic assignment and simulation model runtime.

2.4.2.1 Single Source Shortest Path Algorithms

SSSP algorithms find the minimal cost paths from a single origin vertex ($v \in V$) to all destination vertices ($v' \in V$) within the graph. Within the context of an OD-based assignment, this corresponds to the paths from a single zone (z $\in$ Z) to all other zones (and regular vertices).

These algorithms produce both the route and cumulative cost through the network using two arrays. The route through the network is provided by the *back-edge* or *back-vertex* array, which for each vertex contains the edge or previous vertex with the lowest cumulative cost back to the origin vertex. The *back-cost* array contains the cumulative cost to the vertex from the origin. If there are multiple routes through the network with equivalent cumulative costs then only a single route is stored. In this section, SSSP algorithms are presented as pseudocode inline with how they are commonly presented in research papers [208]–[210]. Equation-based descriptions of these algorithms can be found in the literature [211]–[214].

There are many SSSP algorithms, which may only be applicable to certain types of graph. Many SSSP algorithms are *relaxation* based. The algorithms begin with an approximation of the cumulative cost to each vertex within the graph, which is gradually refined until the minimal cost routes are found. Dijkstra's algorithm [215] is one of the most commonly used SSSP algorithms. It is asymptotically the fastest serial SSSP algorithm for directed graphs with non-negative weights when implemented using a Fibonacci heap [216]. Alternate algorithms or variations to Dijkstra's algorithm may have lower time complexities in other cases, such as Dial's algorithm for graphs with positive integer edge weights [217], or when the graph is guaranteed to be acyclic, in which case topological sorting provides linear time complexity [218]. Dijkstra's algorithm is a work-efficient sequential SSSP algorithm, which uses a priority queue to process the vertices and edges of the graph in an efficient order, reducing the total amount of work to be carried out in most cases. Algorithm 1 describes the base case of Dijkstra's SSSP algorithm. Figure 2.8 provides a directed weighted graph used to illustrate the algorithm. It begins by over-estimating the distance to each unvisited vertex within the graph, and a known cost of 0 to the source vertex (vertex a in Figure 2.8). Starting with the source vertex, the unvisited neighbouring vertices are considered, calculating a new cumulative cost to each connected vertex (vertices b, c & d). Once all neighbours have been investigated, the current vertex is removed from the unvisited set, and it will never be visited again. If the unvisited set of vertices is not empty, the vertex with the lowest cumulative cost is selected and the neighbours investigated (vertex b with a cumulative cost of 1 once vertex a has been removed). This process is repeated when the unvisited set is empty, or there are no more vertices in the unvisited set which are reachable.

An alternative work-efficient SSSP algorithm historically used within macroscopic road network assignment and simulation is the D'Esopo-Pape algorithm [212]. The D'Esopo-pape algorithm uses a double-ended queue of candidate vertices to be be considered, to minimise work required to find the shortest paths. As edges are relaxed, newly encountered vertices are appended to the double ended queue, while previously encountered vertices are prepended to the queue. Although this algorithm has poor worst-case complexity, if often shows good performance due to the order in which vertices are processed [198], [219]–[221]. Algorithm 2 illustrates

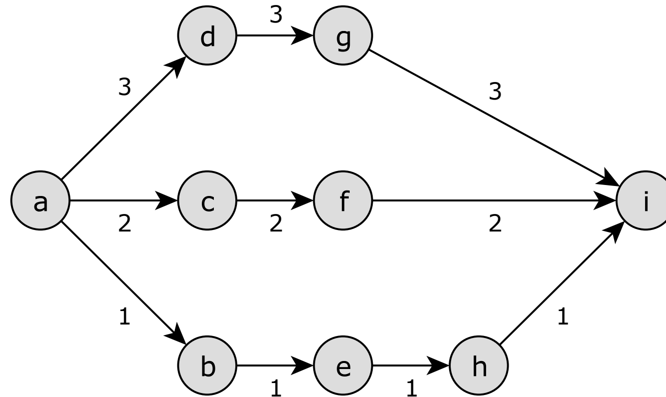


Figure 2.8: A small directed weighted graph containing 9 vertices and and 10 weighted edges.

Algorithm 1 Dijkstra's SSSP Algorithm

Input:

G is Graph with edge costs C,
V is the vertices of graph G,
E is the directed edges of the graph G,
s is the source vertex

Output:

BV is Back-Vertex array of length $|V|$,
BC is Back-Cost array of length $|V|$

1: Let Q be the set of all unvisited vertices
{Initialise the result arrays and data structures}
2: **for each** $v \leftarrow 1$ to V **do**
3: $BC(v) \leftarrow \infty$
4: $BV(v) \leftarrow |V| + 1$
5: $Q.insert(v)$
6: **end for**
7: $BC(s) \leftarrow 0.0$
8: **while** $Q \neq \varnothing$ and $min(BC(v)) < \infty$ **do**
9: $v \leftarrow$ vertex in Q with min $BC(v)$
10: $Q.remove(v)$
{Attempt to re-label each edge leaving the vertex v}
11: **for each** neighbour v' of v **do**
12: **if** v' in Q **then**
13: $e \leftarrow v, v'$
14: **if** $BC(v) + C(e) < BC(v')$ **then**
15: $BC(v') \leftarrow BC(v) + C(e)$
16: $BV(v') \leftarrow v$
17: **end if**
18: **end if**
19: **end for**
20: **end while**

---

the D'Esopo-pape algorithm in detail. Initially, the cumulative cost values for each vertex from the origin are initialised to a large value providing an over-estimate, and the per-vertex edges towards the origin are initialised to an invalid edge value. The cumulative cost for the origin vertex is set to 0, and the queue of candidate vertices is initialised to contain only the origin vertex. An array of values is used to track if a vertex has never been in the queue, is currently in the queue or has previously been in the queue. While the queue of candidate vertices is not empty, a vertex is selected from the head of the queue. The edges departing that selected vertex are iterated, and each edge is relaxed, to determine if a shorter path has been found. If so, the cumulative cost and edge for the destination vertex are updated, and if the destination is not currently in the queue, it will either be appended to the queue if it has not yet been seen, or if it has already been in the queue it will be prepended onto the front of the queue. This process is repeated until the queue is empty. By investigating previously visited vertices before unknown vertices, the algorithm tends to find shortest paths with reasonable efficiency.

Using Figure 2.8 as an example directed graph, to find the shortest paths from source vertex a, the queue of candidate vertices is initialised to only contain the origin a. During the first iteration, the vertex a is selected from the head of the queue, and the departing edges are all explored. As the cumulative costs to each vertex b, c and d were initialised to an invalid edge number (≥ 10 if edges are zero-indexed) all three relaxations will result in updated costs (of 1, 2 & 3 respectively) and routes. As these vertices have not been visited before, they are each appended to the queue of vertices and marked as previously encountered. This iterative process is repeated while the queue is not empty. For this example, vertex i will first be encountered via vertex h during the 5th iteration, setting an initial cumulative cost of 4. On the seventh iteration it would then be re-encountered via vertex f, but the relaxation will not succeed as the new route has a cost of 6.

Work-efficient priority queue based algorithms such as Dijkstra's [215], Dial's [217] and the D'Esopo-Pape [212] algorithms do not present a high degree of parallelism. The order in which the vertices are processed provides the work-efficiency and high performance in CPU-based implementations. Other algorithms which do not target work-efficiency can present higher degrees of parallelism, making them more suitable for many-core parallel processing architectures. The Bellman-Ford algorithm [213], [222] is a much less efficient SSSP algorithm [223] but it presents a high degree of potential for parallelisation.

The Bellman-Ford algorithm is a relaxation-based algorithm, iteratively improving estimates of the cumulative cost to each vertex in the graph from a single origin vertex. Every edge within the graph is considered at each iteration of the algorithm, until the longest possible path containing $|V| - 1$ edges has been considered. By attempting to relax each edge of the graph at each iteration, the shortest path is guaranteed to have been found. Algorithm 3 presents the algorithm. It can be very inefficient for many graphs, as on average the number of edges within

Algorithm 2 D'Esopo-Pape Sequential SSSP Algorithm

Input:

G is Graph with edge costs C,
V is the vertices of graph G,
E is the directed edges of the graph G,
s is the source vertex

Output:

BE is Back-Edge array of length $|V|$,
BC is Back-Cost array of length $|V|$

```
1:  Let Q be the double ended queue of candidate vertices
2:  Let F be an array of flags, 0 if vertex is unseen, 1 if present and 2 if previously seen.
    {Initialise the result arrays and data structures}
3:  for each v ← 1 to V do
4:     BC(v) ← ∞
5:     BE(v) ← |E| + 1
6:  end for
7:  BC(s) ← 0.0
8:  Q.append(s)
9:  P(s) ← 1
    {While the double ended queue is not empty, pop a vertex}
10: while Q ≠ ∅ do
11:    v ← Q.pop()
12:    P(v) ← 2
       {Attempt to relax each edge leaving the vertex v}
13:    for each neighbour v′ of v do
14:       e ← v, v′
15:       if BC(v) + C(e) < BC(v′) then
16:          BC(v′) ← BC(v) + C(e)
17:          BE(v′) ← e
             {Append to Q if unseen, Prepend if known.}
18:          if P(v′) = 0 then
19:             Q.append(v′)
20:             P(v′) = 1
21:          else if P(v′) = 2 then
22:             Q.prepend(v′)
23:             P(v′) = 1
24:          end if
25:       end if
26:    end for
27: end while
```

a graph which will result in a successful relaxation is relatively low. The advantage however is that it presents a high level of data-parallelism, as the many edges can be relaxed concurrently within an iteration. The original Bellman-Ford algorithm has a time complexity of $O(|E||V|)$, however, the complexity can be improved on average through several techniques, by improving the work-efficiency of the algorithm. The number of iterations required within the algorithm can be reduced, from the worst case of $|V-1|$. If there are no successful relaxations within an iteration of the algorithm, then no further relaxations can occur, and the algorithm can be terminated early [224]. The number of attempted relaxations within an iteration can also be reduced, using a vertex-frontier [225], [226]. Based on the vertices which had updated cumulative costs in the previous iteration, the set of edges which may result in updates this iteration can be found, reducing the total work performed per iteration. However, even with these improvements when implemented in serial the Bellman-Ford algorithm is generally outperformed by the more work-efficient algorithms presented.

Using Figure 2.8 as an example directed graph, to find the shortest paths from source vertex a, the base-case Bellman-Ford algorithm would require eight iterations to ensure the the shortest paths have been found. Cumulative costs are initialised to ∞, other than the for the source vertex a. During the first iteration, all ten edges are considered, with only e_{ab}, e_{ac} & e_{ad} leading to shorter routes. All ten edges are considered again during the second iteration, with e_{be}, e_{cf} & e_{dg} resulting in updates. The third iteration leads to 2 edges leading to new lower cost routes via e_{eh} & e_{fi}, with the fourth iteration leading to an update via e_{hi}. For the remaining five iterations no relaxations are successful although the ten edges are once again considered at each iteration.

Work-efficient SSSP algorithms are difficult to parallelise without reducing the efficiency which enables their high serial performance. Task-parallelism is often applied in multi-threaded CPU-based systems, perform the SSSP as independent tasks operating over separate origins. Alternate SSSP algorithms have been proposed as compromises between work-efficient and highly-parallel algorithms, such as the delta-stepping algorithm [208] can offer improved performance in some cases, but still only present a limited degree of parallelism.

GPU accelerated graph-processing libraries provide SSSP implementations such as Gunrock [227], CuSha [228] & NVGraph [229], use algorithms which expose greater levels of parallelism, such as the Bellman-Ford algorithm to leverage the performance available from the many-core GPU architecture. These GPU accelerated implementations can show considerable performance improvements over work-efficient implementations in CPU based libraries, such as the Boost Graph Library (BGL) [230] and the Parallel Boost Graph Library (PBGL) [231], for large-scale dense graphs with low-diameters, such as social network interaction graphs [225]. However, unlike generalised GPU accelerated libraries, macroscopic road network simulation involves the application of the shortest path against low-density, high-diameter graphs characteristic of

Algorithm 3 Original Bellman-Ford SSSP Algorithm

Input:
G is Graph with edge costs C,
V is the vertices of graph G,
E is the directed edges of the graph G,
s is the source vertex
Output:
BE is Back-Edge array of length $|V|$,
BC is Back-Cost array of length $|V|$
{Initialise the Back-Cost and Back-Edge arrays}
1: **for each** $v \leftarrow 1$ to V **do**
2: $BC(v) \leftarrow \infty$
3: $BE(v) \leftarrow |E| + 1$
4: **end for**
5: $BC(s) \leftarrow 0.0$
{Repeatedly relax all edges in the graph}
6: **for** $i \leftarrow 1$ to $|V| - 1$ **do**
7: **for each** $e \leftarrow E$ **do**
8: $v, v' \leftarrow e$ {Source and destination vertices from edge}
9: **if** $BC(v) + C(e) < BC(v')$ **then**
10: $BC(v') \leftarrow BC(v) + C(e)$
11: $BE(v') \leftarrow e$
12: **end if**
13: **end for**
14: **end for**

transport networks. This limits the applicability of existing many-core implementations.

2.4.2.2 All Pairs Shortest Path Algorithms

As an alternate to SSSP algorithms, APSP algorithms compute the shortest paths between all pairs of vertices in the graph ($v, v' \in V$), rather than those for a single origin vertex. Essentially they are equivalent to performing $|V|$ instances of a SSSP algorithm. APSP algorithms therefore have much higher memory requirements than SSSP algorithms, as the output shortest paths edge and cost arrays are $|V|$ times as large, along with potentially a much larger amount of storage being required to perform the algorithm. This can influence the choice of shortest path algorithm to be used, especially for large graphs. In the context of macroscopic road network assignment, APSP algorithms perform more work than is normally needed, and require a significant amount of additional storage. Only the paths between pairs of zones (a subset of all vertices) are required, rather than between all pairs of vertices, and are therefore to offer improved performance compared to the SSSP algorithms in use by current state of the art macroscopic assignment models.

The Floyd-Warshall algorithm [232], Johnson's algorithm [233] and the Pettie 2004 algorithm [214] are APSP algorithms which can be applied to real-weighted graphs which do not contain

negative-weight cycles. The Floyd-Warshall [232] algorithm is an APSP algorithm for directed weighted graphs, without negative cycles. It incrementally improves estimates for the shortest path between any two vertices until the the lowest cumulative cost has been found, for each pair of vertices in the graph. It achieves this with a time complexity of $O(|V|^3)$. The algorithm considers paths which only involve internal vertices which come from a restricted set. The restricted set of vertices begins as the empty set and grows in size by an additional vertex at each iteration. Initially, only single-edge paths between each origin and vertex are known. For the example graph in Figure 2.8 ten single-edge paths are known, such as (a, b), (b, e) or (d, g). After the first iteration paths which traverse the first vertex are considered. This does not result in any updates using the example graph. At the second iteration, paths which traverse the first and second vertex are found (i.e. (a, b, e)). This is repeated until the full set of vertices are considered as intermediate nodes, for each origin-destination pair. For the example graph, this would find the path (a, c, f) during the third iteration, path (a, d, g) during the fourth iteration, paths (a, b, e, h) & (a, b, h) in the fifth iteration and so on.

Johnson's algorithm [233], first published in 1977, uses both the Bellman-Ford and Dijkstra's SSSP algorithms to find the shortest paths between all pairs of vertices in the network. The Bellman-Ford SSSP algorithm is used to ensure that no negative-weight cycles are present in the network, and re-weight the graph to remove negative cost edges. $|V|$ invocations of Dijkstra's algorithm are then applied, to find the shortest paths from each origin. Time complexity of $O(|E||V| + |V|^2 log|V|)$ is achieved, based on the time complexities of the Bellman-Ford algorithm, and that of the Fibbonaci-heap based Dijkstra's implementation. In sparse cases, this algorithm performs fewer comparisons than the Floyd-Warshall algorithm.

More recent algorithms such as the algorithm proposed by Pettie in 2004 offer further advancements over the Floyd-Warshall or Johnson's algorithm in terms of time complexity to $O(|E||V| + |V|^2 log\ log|V|)$ [214], although in the context of road network assignment between zones they perform a significant amount of additional work required.

2.4.3 Macroscopic Road Network Simulation Software

There are multiple commercial macroscopic modelling software packages in used within academia and industry, including PTV Visum [234], SATURN [235] as examples.

PTV Visum [234] is a macroscopic road network simulation tool developed by PTV AG as a part of their comprehensive suite of modelling tools. Visum follows a bi-conjugate Frank-Wolfe algorithm [194] for iterative road network assignment, finding shortest paths using a variation of the algorithm presented by Dibbelt et al. [201], [236]. Parallelism is provided for multi-core CPUs, but reports negligible performance improvements from beyond 8 processor cores [237].

Simulation and Assignment of Traffic to Urban Road Networks (SATURN) is a macroscopic application suite used for the analysis and evaluation of traffic management schemes [235].

Originally developed by the Institute for Transport Studies at the University of Leeds, and released in 1982, SATURN was originally developed as a combined simulation-assignment model. It has since been extended to support both "pure junction simulation" and execution as a "conventional traffic assignment model, with or without simulation" [202]. It contains many applications, ranging from interactive graphical tools for the processing of input and output files, as well as multi-core CPU applications for the modelling and simulation of macroscopic road networks. The multi-core version of the assignment and simulation tool, originally released in 2009, uses task-parallelism to reduce application runtime by computing the shortest paths from several origin zones concurrently as independent tasks using multiple CPU threads, implemented using OpenMP [238]. SATURN is heavily used within the UK transport modelling sector for large scaler regional modelling [239].

2.4.4 Summary

This section has explored macroscopic road network assignment and simulation models. The structure of a typical combined assignment-simulation model such as those used in SATURN or Visum, which follow Wardrop's principle of traffic equilibrium. Additionally, a review of shortest path algorithms which are used heavily during the iterative assignment process is provided, within the context of parallelisation using many-core GPUs. Next, Section 2.5 provides a summary of this chapter.

2.5 Summary

This chapter has explored the forms of transport network simulation focussing on both microscopic road network simulations and macroscopic road network simulations, the use of GPUs for general purpose computing through CUDA, and how this can be applied to the computationally expensive task of road network simulation. Based on this review of literature, several ares of investigation have been identified for investigation within the remainder of this thesis.

It has provided an understanding of how microscopic road network simulations are structured, and the advantages and disadvantages of the techniques compared to alternate approaches. The key models which are fundamental to road network microsimulations have been discussed, such as car following behavioural models and gap acceptance behaviours for junctions. For each type category of model there are many to choose from, which attempt to produce more realistic behaviours, or account for factors such as the environment or different modes of transport. The state of the art road network simulation packages were covered, highlighting Aimsun and PTV Vissim as two simulators which leverage parallelism through multi-core CPUs to attempt to tackle the high computational demand of microscopic modelling. However, none of the widely used simulation tools attempt to leverage GPUs for high levels of performance. There

have been several smaller investigations into GPUs for microsimulation, through techniques such as cellular automata.

Macroscopic road network models were also reviewed. This uncovered the iterative nature of the convergent assignment and simulation models. It was highlighted that the calculation of shortest paths through the transport network accounts for a large amount of the total time spent executing these models. As such, alternate shortest path algorithms were reviewed, and their suitability for parallelisation within road network assignment modelling considered. As with micro-scale simulation, there is little work on the application of GPUs to macroscopic road network simulations in literature. However, GPUs are shown as suitable for shortest path calculations, which could be embedded within macroscopic assignment modelling.

The remaining chapters of this thesis are as follows.

Chapter 3 investigates the application of GPUs to microscopic road network simulations, through the specification of a subset of behavioural models and the definition of a scalable benchmark network which can be used to directly compare the performance of CPU and GPU based implementations. The implementation of the GPU accelerated model through the use of the FLAME GPU ABM framework is described, and experimental benchmarks are carried out to evaluate simulation performance at a range of scales.

Chapter 4 expands on the work of Chapter 3, improving the GPU-accelerated simulation performance through the use of a specialised communication pattern for many-core ABMs for network-based communication. This communication strategy significantly improves the efficiency of message-list based communication for agents which exist within an graph-like environments. An abstract ABM designed to benchmark the communication patterns exhibited by road network behavioural models is also presented.

Chapter 5 presents the application of of GPUs to macroscopic road network simulation and assignment modelling, with a focus on improving the performance of the computationally expensive assignment process. This is achieved through a novel shortest path algorithm for many source vertices, for graphs which exhibit high-diameter and low-radius which are characteristic of road networks. The novel algorithm is embedded within a commercial simulation tool, and the performance evaluated against the serial and multi-core CPU versions of the simulator using real-world transport networks.

Chapter 6 evaluates the relative benefits of GPU acceleration on road network simulations using microscopic or macroscopic techniques. A scalable, artificial benchmark road network defined in Chapter 3 is applied to the macroscopic road network simulation work of Chapter 5 to allow this comparison to be made.

Finally, Chapter 7 provides a conclusion to this thesis, summarising the work carried out and suggesting potential avenues for future work.

Chapter 3

GPU Accelerated Microscopic Simulation

3.1 Introduction

Microscopic simulations of transport networks are fine-grained simulations which model the behaviour of individual vehicles and their interactions with each other and with the environment. These types of simulation can be modelled using Agent Based Model (ABM), where simple behaviours are defined which combine with interactions between individuals and the environment to form the complex emergent behaviours which can be observed in the real world. Unfortunately, this fine level of detail comes with a high performance cost, being much more computationally expensive than higher level methodologies used in transport simulation, such as macroscopic and mesoscopic simulations.

Current tools used in the transport modelling sector which deploy microscopic level simulation, such as Aimsun [122], Vissim [121], MATsim [155] and SUMO [120], can have considerable execution run-times, especially for large scale simulations with simulations executing slower than real-time, with individual simulations potentially taking many hours to complete. This limits the overall effectiveness and uptake of microscopic simulation within the transport sector [14].

Some of these simulators leverage parallel processing in a shared-memory multi-core environment to reduce simulation runtime. The exact parallel methodology varies between tools, using both task-parallel and coarse-grained data-parallel approaches. As discussed in Section 2.2, task-parallelism involves the distribution of independent processing tasks to separate processing threads, whilst coarse-grained data-parallelism applies the same algorithms to different units of data, where the individual units of data are relatively large.

Although the above approaches are well-suited to multi-core Central Processing Units (CPUs), many-core architectures such as Graphics Processing Units (GPUs) can offer much higher levels

of parallelism, and significantly greater levels of raw compute performance. To access the high levels of performance, algorithms and data structures must expose high levels of parallelism and enable good memory-access patterns. Typically fine-grained data-parallelism is used, where the same algorithms are applied to relatively small individual units of data. GPUs have been used previously for microscopic transport network simulation, as both cellular automata [23], [26] and also using multi-agent systems [24]–[26], [240], but general purpose software tools used in industry are yet to adopt GPUs.

This chapter aims to evaluate the suitability of GPUs for microscopic road network simulations using current techniques found within GPU accelerated ABM frameworks, which have previously been demonstrated as appropriate in other domains, such as pedestrian simulations. This is achieved by describing a minimal set of microscopic road network behavioural models and features to support performance evaluation using an artificial, scalable benchmark road network, providing contribution C1 of this thesis. The described set of behavioural models and scalable benchmark network allow the performance of alternate simulation implementations to be compared at various scales and with different levels of vehicle demand. By not selecting a subset of behaviours implemented within commercial tools suitable for the simulation of complex real-world networks, the feasibility of implementing and cross-validating such a model is increased. This builds the foundation on which algorithmic changes and optimisations may be investigated.

The chapter is structured as follows. First, state-of-the-art CPU-based simulators are described in Section 3.2, followed by Section 3.3 providing the definition of a benchmark microsimulation model based on a subset of the models implemented within Aimsun [143], a widely used commercial tool. A scalable benchmark model is proposed and used to benchmark a CPU based simulator in two experiments. Subsequently in Section 3.7 a GPU accelerated implementation of the model is described, using the Flexible Large-scale Agent Modelling Environment for Graphics Processing Unit (FLAME GPU) framework. This model is cross-validated against the reference CPU simulator and the performance is assessed using the previously defined benchmark model. Finally the chapter is concluded in Section 3.8.

3.2 CPU-based Microsimulation

Leading commercial and open source software packages for transport network microsimulation such as SUMO, Aimsun and Vissim use single-core CPU architectures and sequential algorithms, or multi-core CPU architectures with task-parallel and coarse-grained data-parallel algorithms [144], [241]. Tools which leverage parallelism can offer considerable performance improvements compared to sequential applications, reducing the time required for simulations to execute.

In these parallel tools, the work-load of the simulation is distributed as individual tasks

or coarse-grained units of data across the available processing hardware, but, as with many task-parallel and coarse-grained data-parallel multi-core software applications, the performance improvement from each additional processing core reduces as the number of cores and threads is increased. Figure 3.1 shows the application run-time of Aimsun 8.1 for a simulation with a total vehicle demand of 64000 vehicles, with up to 25000 vehicles in the simulated region at one time, as the simulation thread count is increased for multi-core CPU systems. The diminishing returns of additional processor threads are shown, with no significant increases in performance observed beyond six threads. Due to the closed-source, proprietary, nature of Aimsun the specific reasons for these diminishing returns is not known and there are many reasons why a multi-core CPU application may show this type of performance scaling, including cache coherency, parallelisation overheads, workload imbalance or the effects of Amdahl's law. This limits the performance scalability of the application, with large simulations requiring considerable amounts of time to execute even using CPUs with high numbers of processing threads. Additionally, processor threads were not pinned to specific cores for these benchmarks, relying on the operating systems thread scheduler to manage the execution of CPU software threads across the available processor cores. Improved performance may have been observed if thread pinning were enabled.

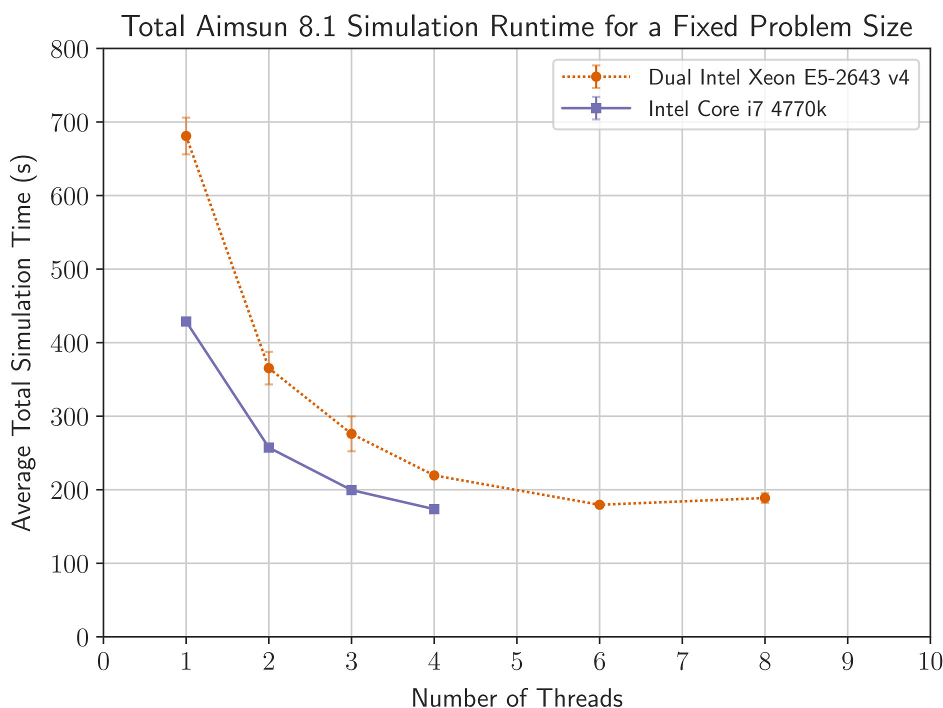


Figure 3.1: The average simulation run-time to complete a one hour simulation a total demand of 64000 vehicles in Aimsun 8.1, for different processor thread counts. The average of three simulations is shown. Error bars show the first standard deviation of the application runtime, but for some data points are smaller than the marker size. As processor thread count increases, total simulation time decreases, but with diminishing returns. Note that the Intel Xeon E5-2643 is a dual-socket system. No performance improvements were observed when using more processing threads than physical cores (Hyper-threading).

3.2.1 Aimsun

Aimsun is a high-performance commercial, multi-core CPU, mesoscopic and microscopic transport simulation package developed by Transport Simulation Systems (TSS). Aimsun can be used to model transport networks of any scale, ranging from a single section of road to a large geographical region and is used globally by governments, consultancies and academic institutions [242]. Aimsun implements a vast array of behavioural models and transport network infrastructure, and is capable of modelling almost any road-network infrastructure and scenario. These features include, but are not limited to:

- Car following model(s)
- Lane changing behaviour
- Overtaking
- Gap acceptance modelling
- Give way modelling and stop signs
- Dynamic traffic infrastructure
- Multiple vehicle classes
- Per-vehicle properties
- Input flows and turning proportions
- Origin-Destination matrices
- Fuel consumption modelling
- Emissions modelling
- Pedestrian simulation
- etc.

Aimsun has a comprehensive graphical user interface and is capable of 2D and 3D visualisation. Full details of the features available in Aimsun can be found in the Aimsun Dynamic Simulators Users' Manual [243].

3.3 Simplified Aimsun Model

To effectively and fairly compare the performance of Aimsun against a GPU implementation the same models and behaviours should be implemented. Aimsun contains such a vast range of functionality that it would be infeasible to implement every feature just to evaluate the performance, so a subset of the models and functionality available in Aimsun 8.1 were selected for implementation.

The models selected provide core functionality for the simulation of road networks and produce data which enables cross-validation of the model implementation against the reference application, but also selected in such a way that other features could be disabled through various

settings within Aimsun, to minimise the performance impact of models and functionality which were not implemented. Table 3.1 details models and features which were selected from the partial list of Aimsun features in Section 3.2.1.

Aimsun Functionality	**Selected for the Simplified Model**
Car following model(s)	Selected, based on Gipps' car following model [128], [143]
Lane changing behaviour	
Overtaking	
Gap acceptance modelling	Selected, used for give-way modelling only
Right of way modelling	Selected, including yellow-box modelling
Give way and stop signs	Selected,
Dynamic traffic infrastructure	Virtual Detectors only, which provide periodic statistics on the movement of vehicles through the detection range
Multiple vehicle classes	1 Class of vehicles selected
Per-vehicle properties	Selected, only properties required for selected models
Input flows and turning proportions	Selected, with constant vehicle arrival and virtual queues
Origin-Destination matrices	
Fuel Consumption modelling	
Emissions modelling	
Pedestrian simulation	

Table 3.1: Models selected for the Simplified Aimsun Model from the partial list of aimsun models and features. Aimsun publications and Documentation provide detailed descriptions of these functionalities ([143], [144], [243])

These features and models were selected to produce comparable behaviour for the procedurally generated grid road network described in section 3.4, without the complexities of modelling the broad range of road network infrastructure present in real world road networks. In general, the specific model chosen to model an observed behaviour is not too important, but by using the same model as implemented in the reference CPU simulator evaluation can be more direct. Further details of the selected behavioural models can be found in section 2.3.2 and within the Aimsun documentation manual [144], [243].

3.4 Benchmark Network

To evaluate the performance of implementations of the simplified Aimsun model described in section 3.3 at various scales, an artificial network is described which unlike real-world transport networks can be scaled to larger and larger sizes.

The benchmark transport network is a 2D grid of single-lane one-way roads, with adjacent rows and columns of the grid network traversing in opposite directions. This is inspired by the grid road networks of the world such as the famous Manhattan street grid. Junctions are controlled through the use of stop-signs at each junction-entry, and each junction is defined as a yellow box junction to reduce the risk of grid-lock. Figure 3.2 shows the arrangement of road

sections and junctions which form the grid road network. The arrangement of turns from one section of road to another within a junction is also shown.

To allow high level comparisons between alternate implementations of the microscopic model, and collect information on the movement of vehicles within the simulated network, three virtual transport network detectors are placed on each section of the road. The detectors are placed offset from the start, in the centre, and offset from the end of each section of road.

The *network grid size* can be used to control the scale of the generated grid-based network, and it can be used to calculate the numbers of sections, junctions and turns in the network. A network with *grid size* (G) contains G^2 junctions, $2G(G+1)$ road sections, $4G^2$ turning sections, $2G$ entrances and $6G(G+1)$ detectors. Table 3.2 describes the parameters involved in network generation.

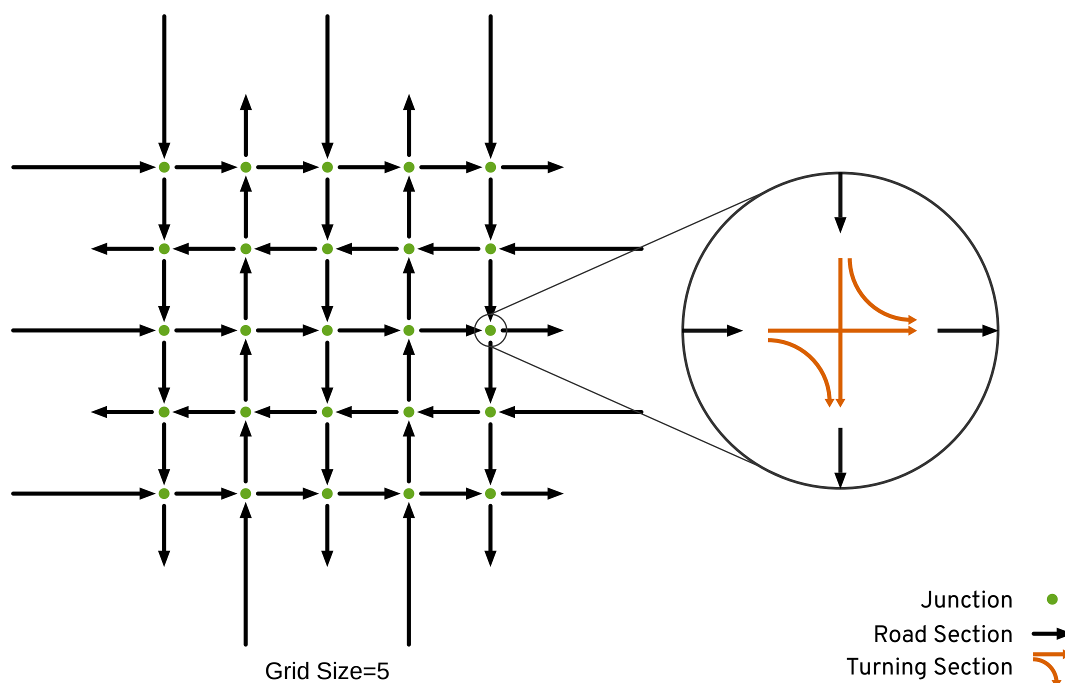


Figure 3.2: A 5×5 example of the procedurally-generated artificial grid network, showing the overall structure of the network and the arrangement of turning sections within a junction. The network can be scaled to any size, with networks of up to 576×576 used during benchmarking.

Parameter Name	Description
Grid size (G)	The number of junctions for each row and column of the grid
Section length	The length of each road section between junctions in metres
Junction length	The length or diameter of junctions within the network
Exit turn proportion	For junctions with 1 exit destination edge, the proportion of vehicles turning onto the exit is reduced to maintain higher vehicle populations within the simulated road network.

Table 3.2: Parameters for generation of the procedurally generated Manhattan-style grid road network

3.5 Benchmark Experiments

Two sets of benchmark experiments are used to evaluate the impact of scaling the size and density of the transport network on simulation runtimes, using the model described in section 3.3 and the artificial network described in section 3.4: the Grid-scale experiment; and Input-flow experiment.

3.5.1 Grid-Scale Experiment

To evaluate the performance impact of total problem scale, the *Grid Size* of the procedurally generated network is varied from 2 to 576. This results in larger populations of vehicles, a greater number of vehicle detectors and a larger transport network. As vehicles are only inserted from the edges of the network, the ratio of vehicle input roads to the number of roads is $2G : 2G(G+1)$.

This experiment shows how the performance of the simulator and hardware copes with larger and larger transport networks, i.e. is the simulator capable of city-scale, region-scale of even national-scale simulations. The model and network parameters used for the grid-scale experiment are described in Table 3.3.

Parameter name	Value	Units
Simulation Time	3600	seconds
Time step	0.8	seconds
Reaction time	0.8	seconds
Stop reaction time	1.2	seconds
Detector period	600	seconds
Input flow scale factor	1.0	%
Seed	Random value	Long integer
Section length	1000	m
Junction length	10	m
Exit turn proportion	5	%
Grid size	2 - 576	
Input flow	600	
Detectors per section	3	

Table 3.3: Network and Model parameters used for the Grid-Scale microscopic experiments. Parameters such as the time step, reaction time and detector period were selected as values commonly used within microscopic modelling in the UK as informed by the Aimsun developers.

3.5.2 Input-Flow Experiment

Demand on transportation networks is increasing globally [1]. To evaluate the impact of this on microscopic transport network simulation, the demand on the transport network was varied for several fixed sizes of network. This shows how the simulators performance characteristics scale with higher and higher density of vehicles, which corresponds to the increasing demand on existing transport networks observed in the real world. The model and network parameters used for the input-flow experiments are described in Table 3.4.

Parameter name	Value	Units
Simulation Time	3600	seconds
Time step	0.8	seconds
Reaction time	0.8	seconds
Stop reaction time	1.2	seconds
Detector period	600	seconds
Input flow scale factor	1.0	%
Seed	Random value	Long integer
Section length	1000	m
Junction length	10	m
Exit turn proportion	5	%
Grid size	64, 128 & 256	
Input flow	100 - 1000	
Detectors per section	3	

Table 3.4: Network and Model parameters used for the Input-Flow microscopic experiments. Parameters such as the time step, reaction time and detector period were selected as values commonly used within microscopic modelling in the UK as informed by the Aimsun developers. Multiple grid sizes were used to assess the performance impact of the input flow parameter at multiple network scales.

3.6 CPU Benchmark Results

The experiments described in Section 3.5 were executed using Aimsun 8.1 executed on an Intel Core i7-4770k (4 cores, 8 thread) with 16GB of memory available using Windows 10.. This provides a baseline for GPU implementations to be compared against. Each simulation was repeated 3 times to provide average simulation runtimes. To ensure fair comparisons, working in conjunction with the Aimsun developers, a modified version of Aimsun 8.1 was produced which records the runtime per-simulation iteration, to enable more fine-grained performance information.

Figure 3.3 shows the average simulation runtime for 3 repetitions of each simulation within the Grid-Scale set of benchmark experiments. For small scale benchmarks, Aimsun 8.1 shows short simulation runtimes, with a real-time-ratio (RTR) (the ratio of simulated time against the duration required for the simulation to execute) of 192 for the grid size 8 network containing up to $8,000$ vehicles and 432 detectors. As the scale of the network is increased, the performance degrades with a non-linear relationship to the grid size parameter. The largest simulation which completed faster than real-time contained up to $384,000$ vehicles and $295,680$ km of road, with a grid size of 384. The largest simulation which would complete in Aimsun 8.1 had a grid-size of 512, containing up to $512,000$ vehicles and $1,575,936$ road network detectors. This simulation completed in an average time of 5447.7 seconds, at a RTR of 0.66.

Figure 3.4 shows the average simulation time for Aimsun 8.1 from for 3 repetitions of each simulation at the various scales. At each grid scale (shown by the different series) the multi-core CPU simulation results show relatively linear performance as the input flow is scaled for each fixed size network. As the input flow is increased, the average vehicle density will also increase in conjunction with the total number of vehicles. The benchmark network contains $2G$ entrances, so the total vehicle count will scale linearly with input flow for a fixed network size, however, vehicle density increases near the entrance sections as it takes time for agents

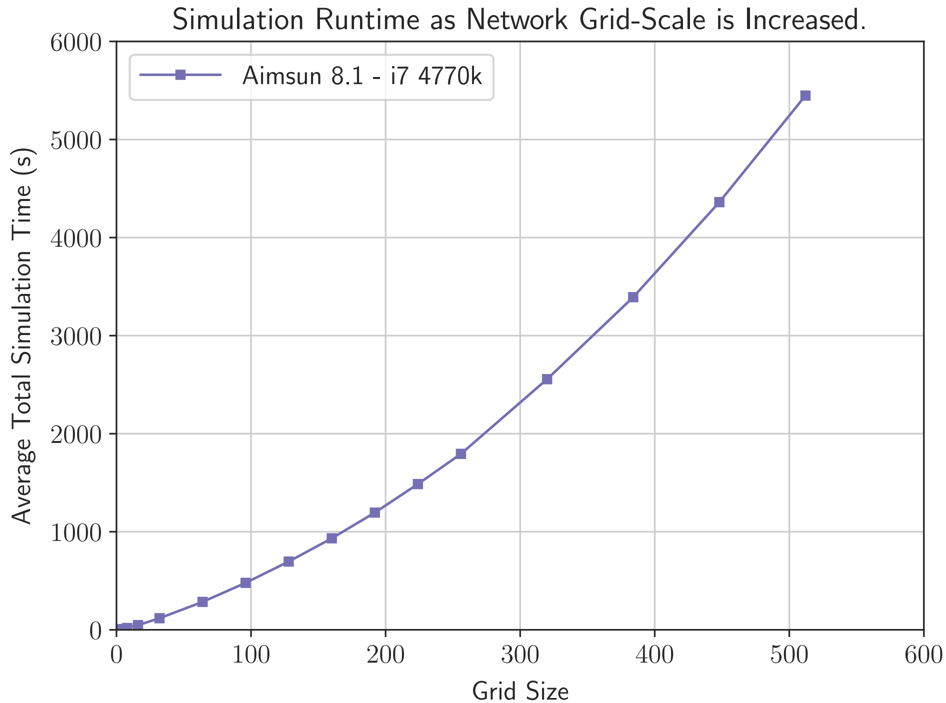


Figure 3.3: Total simulation time for Aimsun 8.1 as the total scale of the simulation is increased. Values shown are the average from 3 repetitions. The model and network parameters used for this benchmark are shown in Table 3.3.

to disperse throughout the network leading to local areas of higher density. The mainly linear relationship between input flow and simulation runtime suggests the Aimsun multi-core CPU implementation is not significantly effected by local density hot-spots.

Further insight into the performance of the simulator can be gained from the per-iteration run-time. Figure 3.5 shows the per-iteration performance of the Aimsun 8.1 simulation for a grid-size of 128. As the simulation progresses the population of vehicles in the simulated region increases. This leads to increasing time required to compute each time step of the simulation, with a linear relationship between iterations due to the linear increase in population size of the periodic vehicle input used by the benchmark network.

Every 750 iterations a small cluster of iterations have increased step times. This periodic effect is correlated with the behaviour of the detectors in the benchmark network, which perform aggregate operations every 10 minutes of simulated time (750 iterations with a time-step of 0.8 seconds). The aggregate operations involved will require additional processing time, and the Aimsun implementation may also write data to disk at this time, leading to the short-term increase in run-time. Even if the output of data to disk is performed asynchronously, it can still have a minor impact the performance of the simulation.

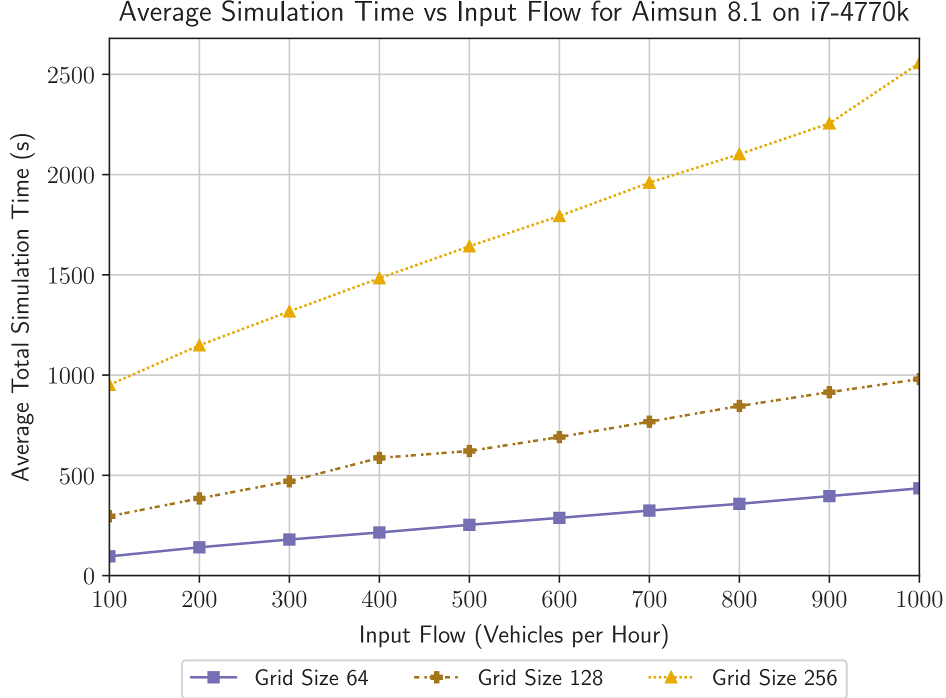


Figure 3.4: Total simulation time for Aimsun 8.1 against input flow for procedurally generated road networks of grid size 64, 128 and 256. Run times shown are the average from 3 repetitions. The model and network parameters used for this benchmark are shown in Table 3.4. Transport for London guidance suggests maximum design capacities of between 900 and 1600 vehicles per hour per lane for urban, single carriageway, one-way roads with speed limits of 30mph ($\sim$ 50kmph) [244]. Benchmarks with input flows greater than 1000 vehicles per hour were not performed due to stop-sign induced queues preventing vehicles from entering the simulated region of the artificial benchmark road network which occurred more frequently as input flows were increased.

3.7 GPU Microsimulation

As discussed in Section 2.1, ABM can be used to model microscopic simulations of transport systems. Many software frameworks for ABM exist which abstract the complexities of parallel processing away from the modeller. D-Mason [166], Repast HPC [20] and FLAME II [169] use distributed or CPU-based task-parallelism or coarse-grained data-parallelism to offer improved performance compared to single threaded applications. Yet few provide support for performing high performance simulations using many-core processors such as GPUs. One framework which does provide GPU accelerated ABM is FLAME GPU [179].

3.7.1 FLAME GPU

FLAME GPU is a GPU accelerated ABM framework, which uses the CUDA (Compute Unified Device Architecture) API for NVIDIA GPUs and a state-based representation of agent dynamics to leverage the high levels of performance from the GPU. The complexities of the CUDA programming model are abstracted away from the end-user, enabling high performance simulations without explicit knowledge of the parallel processing model [178], [179], [181].

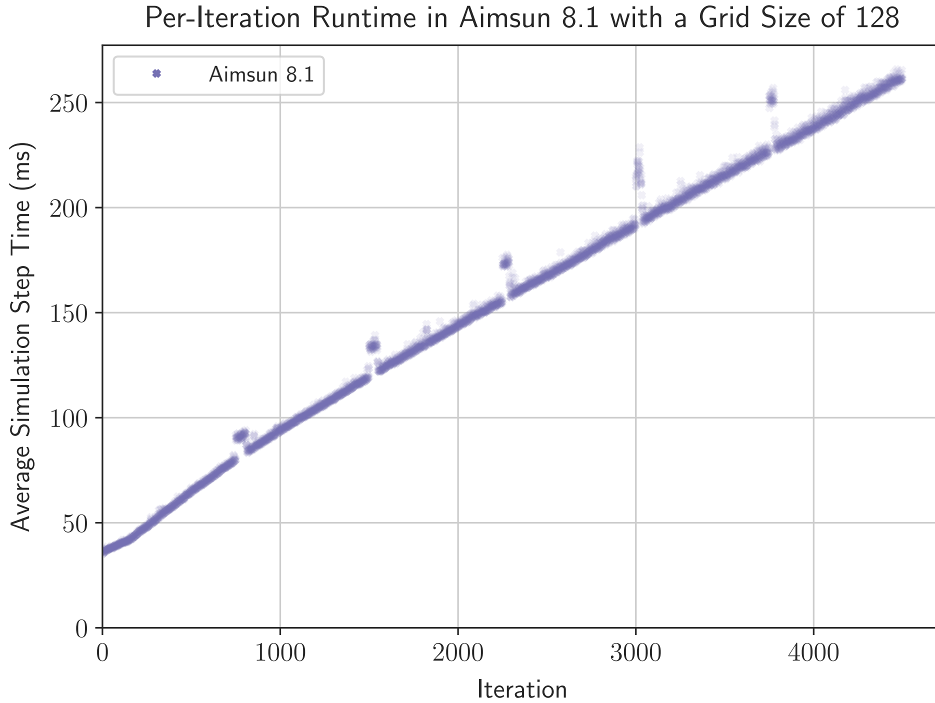


Figure 3.5: Per-Iteration simulation time for a model with grid size 128 and input flow 500 vehicles per hour.

FLAME GPU has been used in multiple modelling domains including but not limited to pedestrian crowd simulation [182], cellular simulations [178], [183], Cellular Automata (such as the Game of Life) and flocking models [22], [245]. However, the use of FLAME GPU for road network simulation has been limited.

FLAME GPU simulations make use of a state-based representation of agent dynamics to simplify development and enable key performance optimisations on behalf of the modeller. Communication is achieved through message-lists, the output and input of which also describe dependencies within the state-based representation of the model. By using explicit message lists for communication rather than allowing direct access to agent memory, race conditions (a common pitfall within parallel processing) are prevented during agent communication, by ensuring that agents may update their local properties without altering the contents of a message being accessed by another agent.

At each step of a FLAME GPU simulation, the model is formed of many agent functions split into *execution layers*, which are executed sequentially. Within an execution layer agent functions should be independent and therefore able to operate concurrently.

In large scale FLAME GPU models, the communication between individuals often has a significant impact on the performance of the simulation [29]. Message lists can therefore be accessed using specialised message partitioning techniques to reduce the scale of message lists each agent must iterate, increasing performance where appropriate. Within FLAME GPU 1.4, there are 3 methods of communication: all-to-all communication; discrete partitioned messaging;

and spatially partitioned messaging for agents in continuous space.

All-to-all communication involves each agent parsing messages from all other agents. Discrete Partitioned Messaging is a specialisation for agents which exist in discrete space and communicate with local neighbours, such as Cellular Automata. Spatially Partitioned messaging allows communication with all agents within a fixed radius of the individual, with the partitioning scheme requiring a radius and environment bounds. The continuous 2D or 3D environment is divided into a grid of cells, where individuals receive messages from their cell and the moore-neighbourhood of cells [179].

Of these partitioning schemes all-to-all communication and spatially partitioned messaging are of interest for road network microsimulation, with performance implications. Non partitioned messaging is computationally expensive ($O(n^2)$ message iteration loop when reading messages) but has little overhead cost. Spatially partitioned messaging generally provides a performance improvement, as each agent processes fewer messages. However, there can be significant overhead cost for the construction of data structures used to efficiently access the appropriate messages. Care must also be taken to ensure that the communication radius is sufficiently large to not influence the models implemented.

3.7.2 FLAME GPU Implementation of the Gipps' Car Following model

Early during the development of the simplified Aimsun model, a further simplified version of the model was implemented which only contained the Gipps' Car Following model, with random route decisions at junctions and no collision avoidance models implemented. This very simple model was used to evaluate the suitability of this model for the GPU and the impact of communication on the car following model, and benchmarked on a very simple grid network, not too dissimilar from that described in Section 3.4.

This model only contained vehicle agents, and accessed a transport network stored in CUDA constant memory (via FLAME GPU's Simulation Constants). This does limit the scale of the network able to be simulated due to the limit of 64kB of constant memory being available in CUDA devices. This was changed for later implementations.

In the naive model, at each simulation step vehicles output their observable properties to a message list. A subsequent agent function iterates each message from the message list to find the information about the preceding vehicle, required for Gipps' car following model. Agents which have reached the end of their current segment of road randomly select a new segment from the roads connected at their current junction. The performance of these simulations were evaluated using all-to-all communication and spatially partitioned messaging using several partitioning radii, for simulations where the population is varied for a fixed size network.

These benchmarks were implemented using FLAME GPU 1.4 for CUDA 7.0, and executed on an Intel Core i7 4770K workstation using a NVIDIA TESLA K20c GPU. A fixed grid network

containing $N = 16$ sections of road, each $10000m$ in length is used, and the number of agents in the simulation is varied from 2^8 up to 2^{18}. The benchmark simulations are repeated using both all-to-all and spatially partitioned communication. Figure 3.6 shows the per-iteration simulation performance against the number of agents, where performance has been measured by averaging the simulation time over 100 iterations. This shows that at very small populations brute force communication offers higher performance, but as the population size grows spatially partitioned offers improved levels of performance. Lower partitioning radii show higher performance.

Figure 3.7 shows the average agent iteration performance (calculated as average iteration time / population size). This shows that for smaller populations of individuals, as the population is increased the per-agent performance improves, as hardware utilisation increases. Once the population, and therefore number of processing threads, is sufficiently large to fully utilise the device, the per-agent population decreases as population grows. This can be attributed to agents having to parse larger and larger message lists which must be parsed by each agent. Non-partitioned messaging shows significantly lower per-agent performance as the scale of message lists grows at a higher rate than spatially partitioned messaging within this type of network, with relatively uniformly distributed agents.

Although this shows that spatially partitioned messaging does improve performance, it highlights the need for an appropriate messaging partitioning strategy.

3.7.3 FLAME GPU Implementation of the Simplified Aimsun Model

The model described in Section 3.3 has been implemented using FLAME GPU. The models contains two agents types, for Vehicles and Detector Agents. Vehicle agents use several states to distinguish between agents which must perform different behaviours. Detectors only use a single state.

Figure 3.8 shows the state machine for the vehicle and detector agents implemented in this model. The state machine represents the process followed by each agent at each step of the simulation, highlighting the relationship between agent states, functions and message-lists. Agents begin the iteration in one of the possible states for the agent type. Agent functions allow agents to switch from one state to another (or remain in the same state). The order in which these functions are executed are controlled by the execution layers. Message lists used for communication highlight dependencies between agent functions. A message list must be written to in an earlier layer than it is accessed, and are reset between subsequent iterations. Vehicle agents which exit the simulation are determined to be *dead* and are excluded from further steps of the simulation.

Agents representing vehicles can be in one of three states: *Queued*, *Road* or *Junction*. The virtual queue of vehicles waiting to enter the simulated region are represented by the *Queued* state of car agents. Newly created *Car* agents generated from a method executed every iteration

in serial on the host are initially placed in to the *Queued* state. Within each simulation iteration, the series of agent functions which occur for Queued state agents implement the constant or exponential vehicle arrival algorithm in the benchmark microsimulation model. The arrival algorithm is implemented to match the behaviour of the reference simulator Aimsun [144], ensuring that vehicles arrive in the order that they joined the queue, and that there is sufficient headway on the simulated road section for the vehicle to enter the network, with an appropriate arrival velocity.

The *Road* and *Junction* states are used to separate the behaviours of agents which are traversing a section of road, and those approaching or traversing a junction. This is to reduce branching and divergence within the agent function, and the use of the two graphs which describe the road network data structure. Vehicles in the Road state exhibiting primarily the car following behavioural model, and interaction with stop signs as they approach junctions. Agents in the Junction state execute the gap acceptance give-way model and yellow-box model used for this study, and subsequently if movement is allowed the vehicle will use the car following model to traverse the junction.

Detector agents are used to represent real-world vehicle detection systems such as induction loops placed within road network infrastructure. The agents follow a much simpler, linear state machine than car agents. These are used to measure traffic conditions, the output of which can be used for interactive behaviour with transport network infrastructure, simulation calibration or simulation validation. The state diagram for detectors is linear, containing three agent functions, each in separate layers. Each function takes input from a message lists output to by vehicle agents during earlier layers within the iteration (the Junction message list, Road message list and Exit message list) to collect and aggregate statistics about individual vehicles which are within or traverse the detector region of effect. Periodically at the end of the iteration a FLAME GPU step function executes, to store the aggregate information from detector agents and reset counters for the next time period.

The transport network is represented using a pair of graphs, represented using the Compressed Sparse Row (CSR) data structure previously described in Section 2.1.5. The first graph represents the roads of the transport network as edges, and the junctions as vertices. The second graph models the connections between roads within each junction , with the each turn represented as an edge, and vertices mapping to the edges connected to the junction. Separate graphs are used to simplify the specification of the road network, and allow the use of FLAME GPU states to minimise the cost of divergence within the agent functions.

3.7.4 Cross Validation

For the performance evaluation of the two simulators to be comparable it is important that software produces comparable results. These simulations are stochastic, so multiple runs using

different random number seeds must be used to collect meaningful values for validation. A set of validation networks were produced targeting specific features and behavioural models, which are described in Table 3.5.

These models were then executed multiple times using both Aimsun 8.1 and FLAME GPU. The same model parameters were used for both implementations, to ensure comparable results, and the relevant output data compared. Stochastic behaviour was minimised where possible to simplify the validation procedure. For instance, vehicle properties which are designed to be sampled from a distribution were applied uniformly across the population when possible. The same parameters were used with both simulation tools. The parameters used for non-stochastic simulations are shown in Table 3.6, and stochastic parameters shown in Table 3.7.

Feature	**Purpose**	**Description**
Constant Vehicle Input	Ensure that vehicles arrive in the simulation correctly, for *constant* input flows	A single 100m edge, with input flows of 100, 200, 400 and 800 vehicles per hour, using deterministic parameters
Car Following Behaviour	Ensure that deceleration and acceleration of vehicles is applied correctly	Two 100m or 500m edges, with a single 10m junction including a stop-sign. Vehicle input flow set to 500 vehicles per hour, using deterministic parameters
Junction Behaviour	Ensure vehicles follow the same junction behaviour for a single junction from the Manhattan style grid	Two 100m input sections, two 100m exit sections, two straight ahead turns of 10m, two 90° turns of 7.85m. Turning proportions of 100% ahead, 100% 90° turn and $50\% - 50\%$, using deterministic parameters
Deterministic Grid Model	Compare behaviour of the Manhattan grid network, where stochasticity is removed	Grid Sizes 2, 4, 8 & 16, simulated for 1 hour with deterministic parameters and 100% ahead turns, using deterministic parameters
Stochastic Grid Model	Compare behaviour of the Manhattan grid network, with stochasticity	Grid Sizes 2, 4, 8 & 16, simulated for 1 hour with stochastic parameters and 50%50% turns, using stochastic parameters

Table 3.5: Details of the networks used for cross-validation.

Parameter Name	**Value**	**Units**
Length	4.0	m
Clearance	1.0	m
Speed Acceptance	1.0	
Sensitivity Factor	1.0	
Maximum Desired Velocity	22.2222	ms^{-1}
Maximum Acceleration	3.0	ms^{-2}
Normal Deceleration	4.0	ms^{-2}
Maximum Deceleration	6.0	ms^{-2}

Table 3.6: Vehicle model parameters used for the deterministic cross-validation of the FLAME GPU and Aimsun models.

The results of each validation model show:

- Constant vehicle arrival is correct, with matching input counts and flow across the network for the *Constant Vehicle Input* validation model (Table 3.8).
- Free-flow car following behaviour is shown to be correct by the average speed of vehicles in

Parameter Name	Mean	Deviation	Minimum	Maximum	Units
Length	4.0	0.5	3.5	4.5	m
Clearance	1.0	0.3	0.5	1.5	m
Speed Acceptance	1.1	0.1	0.9	1.3	
Sensitivity Factor	1.0	0.0	1.0	1.0	
Maximum Desired Velocity	30.5556	2.7778	22.2222	41.6667	ms^{-1}
Maximum Acceleration	3.0	0.2	2.6	3.4	ms^{-2}
Normal Deceleration	4.0	0.25	3.5	4.5	ms^{-2}
Maximum Deceleration	6.0	0.5	5.0	7.0	ms^{-2}

Table 3.7: Truncated normal distributions for vehicle parameters used for the stochastic cross-validation of the FLAME GPU and Aimsun models.

the Constant vehicle arrival models (Table 3.8). The Car Following Behaviour validation network highlights a discrepancy in the number total flow through the network, of 0.2%, with one extra vehicle still in the simulated region (Table 3.9). Further investigation revealed that the velocity of a vehicle approaching a stop sign is correct, as shown by Figure 3.9 and table 3.10, and that the reduced average speed within the simulation is related to interaction with stop signs within a queue, or with the not-implemented gap acceptance model.

- Turning proportions are correctly demonstrated for the three test sets of 100:0, 0:100 and 50:50, within acceptable limits imposed by the use of a random number generator, however, the flow is once again not an exact match (Table 3.11).
- The deterministic and stochastic models show comparable *input flow* for each network, but the *output flow* and final population of the simulation vary by increasing amounts (Tables 3.12 and 3.13). This is accounted for by the increase in the number of junctions as the grid size is increased, which exacerbates the differing behaviour for junctions.

Target Input Flow	Aimsun				FLAME GPU			
	Flow	Input Count	Input Flow	Average Speed	Flow	Input Count	Input Flow	Average Speed
100	98.000	100.000	100.000	13.889	98.000	100.000	100.000	13.889
200	196.000	200.000	200.000	13.889	196.000	200.000	200.000	13.889
400	392.000	400.000	400.000	13.889	392.000	400.000	400.000	13.889
800	784.000	800.000	800.000	13.889	784.000	800.000	800.000	13.889

Table 3.8: Key data for the Constant Entrance Flow validation models

Section Length	Aimsun				FLAME GPU			
	Flow	Vehicles Inside	Vehicles Outside	Average Speed	Flow	Vehicles Inside	Vehicles Outside	Average Speed
100	497.0	3	497	11.812	496.0	4	496	11.397
500	489.0	11	489	11.879	488.0	12	500	11.506

Table 3.9: Key data for the Car Following Behaviour validation models

Iteration	Aimsun Speed	FLAME GPU Speed
1	50.000	50.000
2	50.000	50.000
3	50.000	50.000
4	50.000	50.000
5	50.000	50.000
6	50.000	50.000
7	50.000	50.000
8	42.415	42.415
9	32.432	32.432
10	22.897	22.897
11	14.112	14.112
12	6.697	6.697
13	1.805	1.805
14	0.141	0.141
15	0.000	0.000

Iteration	Aimsun Speed	FLAME GPU Speed
16	0.000	0.000
17	3.415	3.415
18	9.563	9.563
19	17.686	17.686
20	26.277	26.277
21	33.881	33.881
22	39.718	39.718
23	43.729	43.739
24	46.304	46.304
25	47.861	47.861
26	48.777	48.777
27	49.306	49.306
28	49.607	49.607
29	49.778	49.778

Table 3.10: Velocity for the first vehicle in the 100m, 2250 vehicles per hour input flow, Car Following Behaviour validation model.

Turning Proportion		Aimsun			FLAME GPU		
% Ahead	% 90° Turn	Flow	Vehicles Inside	Vehicles Outside	Flow	Vehicles Inside	Vehicles Outside
100	0	993.0	7	993	990.0	10	990
0	100	994.0	6	994	992.0	8	992
50	50	993.0	7	993	990.0	10	990

Table 3.11: Key data for the Turing Proportion validation model.

	Aimsun				FLAME GPU			
Grid Size	Input Flow	Flow	Vehicles Inside	Vehicles Outside	Input Flow	Flow	Vehicles Inside	Vehicles Outside
2	2000.0	1868.0	132	1868	2000.0	1910.0	90	1910
4	4000.0	3466.0	534	3466	4000.0	3718.0	282	3718
8	8000.0	6417.0	1583	6417	8000.0	7012.0	988	7012
16	16000.0	11397.0	4603	11397	16000.0	12210.0	3790	12210

Table 3.12: Statistical summary data for the Deterministic Grid validation simulations.

	Aimsun				FLAME GPU			
Grid Size	Input Flow	Flow	Vehicles Inside	Vehicles Outside	Input Flow	Flow	Vehicles Inside	Vehicles Outside
2	2000.0	1786.0	214	1786	2000.0	1903.0	97	1903
4	4000.0	3421.0	579	3421	4000.0	3606.0	394	3606
8	8000.0	6304.0	1696	6304	8000.0	6801.0	1199	6801
16	16000.0	11360.0	4640	11360	16000.0	12447.0	3553	12447

Table 3.13: Statistical summary data for the Stochastic Grid validation simulations.

3.7.5 GPU Benchmark Results

The experiments described in Section 3.5 were executed using the FLAME GPU 1.4 based implementation of the model described in Section 3.3. FLAME GPU simulations were executed using NVIDIA Titan V GPU (5120 core, 12GB memory), and using a single thread on an Intel Core i7-6850K for CPU-based aspects of the GPU-accelerated simulation. Each simulation was repeated 3 times to provide average simulation runtimes. The time taken for each simulation is logged using FLAME GPU's instrumentation functionality. The GPU-based benchmark results

are compared against the Aimsun multi-core CPU results presented in Section 3.6 which were generated using 4 cores and 8 threads on an Intel i7-4770k CPU.

Figure 3.10 shows the average simulation runtime for 3 repetitions of each simulation within the Grid-Scale set of benchmark experiments, using the alternate communication strategies available in FLAME GPU.

For small scale benchmarks FLAME GPU 1.4 on a modern GPU shows lower levels of performance than that of the Aimsun 8.1 CPU results, using either of the existing communication strategies. At these scales the GPU is not provided with enough work and is underutilised, resulting in the overhead costs outweighing any benefits from executing on the GPU. As the scale of the network is increased, brute force communication becomes more and more computationally expensive, as each agent is processing larger and larger message lists. This quickly becomes slower than the efficient CPU simulator and slower than real time. Simulations which use spatially partitioned messaging shows better scaling, as message lists do not grow as quickly. This yields better performance than the CPU simulator once the total problem size, and therefore agent population is sufficiently large. However, as the problem continues to scale to larger populations, the quantity of messages processed by each agent increases on average as the average density increases. This leads to a loss of performance and ultimately slower than real-time simulations which are also slower than the reference CPU simulations.

Using the brute-force approach of all-to-all communication, the GPU does not offer improved performance over the CPU. This communication approach is highly inefficient for the transport network behaviours where only information from neighbouring vehicles are required. As each individual is processing each message, at multiple stages in each iteration the overall performance impact of communication is significant, outweighing the benefits from using the GPU. Spatially partitioned communication is better, performing faster than the CPU simulator at mid-range scales, where the parallelism outweighs the inefficient communication.

Figures 3.11 to 3.13 show the average simulation time for 3 repetitions of each simulation at three scales of fixed size network, with varying input flows per entrance edge of the simulation. This allows the effect of vehicle density to be evaluated. The individual series in each figure compare the average simulation runtime for each implementation, comparing Aimsun 8.1 on the CPU with FLAME GPU on an NVIDIA Titan V, using both all-to-all and spatially partitioned communication. For the smallest of the three grid sizes, 64, the GPU implementation demonstrates performance advantages compared to the CPU implementation at low input flows values, using using both communication strategies. As the vehicle density increases within the simulation, due to the increased input flow per entrance for a fixed size network, the GPU runtimes increase at a faster rate than the CPU implementations, ultimately taking more time to complete the one hour simulations. The spatial communication pattern shows reduced performance compared to the all-to-all communication strategy at this scale. For the grid size

of 64, the GPU is underutilised as there are not enough vehicles within the simulation. This highlights the additional costs of the more complex spatial communication pattern.

For the medium of the three grid sizes, 128, a similar pattern is observed. However, the GPU implementations outperform the CPU implementations until input flows of 700 or greater. Additionally, the spatial communication pattern outperforms the all-to-all communication until larger input flow values are reached. For the largest of the three grid sizes, 256, the brute force communication pattern outperforms the spatial messaging at almost all scales, and outperforms reference CPU results at all scales.

The performance of the GPU implementations does not scale as well with input flow and therefore vehicle density as well as the reference CPU simulator. This is mainly due to the increase in the size of message lists, which are used to avoid race conditions in the parallel implementation. For global, all-to-all communication, the total message list size has a direct impact on performance. As more vehicles are entering the simulations for larger input-flows, the number of messages that each agent must read, and the number of agents increase, contributing towards the increased runtimes. When using spatially partitioned messaging, as vehicle density increases the average number of messages each agent must iterate increases. Although the spatial partitioning reduces the size of message lists, the runtime increases as the simulations become denser, becoming slower than the all-to-all communication. In part this is due to the large communication radii which must be used to ensure the correct information is found at all vehicle speeds, but also due to the increased cost of building the data structure used to access messages based on spatial location and increased costs of accessing messages through this data structure. For the simulations using the largest size grid (256), the GPU-based simulation using all-to-all communication shows improved performance compared to the reference simulator at all input flow values. At this larger grid scale, the population of vehicles being simulated is large enough that the highly parallel GPU can be fully utilised, overcoming the overhead costs of using the GPU. Meanwhile the multi-core CPU implementation is struggling compared to lower grid scales where the vehicle demand is much lower, even at the lowest of input flows.

3.7.5.1 Per-Iteration Runtime

The per-iteration run-time of the simulations can provide further insight into the relative performance difference between simulation implementations. Figure 3.14 shows this for simulations with a grid-size of 128. CPU results are repeated from Figure 3.5, while GPU results were generated using the spatially partitioned messaging variant of the FLAME GPU implementation, executed on a NVIDIA Titan V GPU. The CPU results from Aimsun show a broadly linear relationship between iteration number and iteration execution time as the simulation progresses, and the population of simulated vehicles increases. The GPU implementation using local communication shows a per-step simulation time initially, while message lists are relatively

small due to the low vehicle density. This may be influenced by additional features within the commercial simulation package which could not be disabled. As the simulation progresses, the per-iteration runtime begins to grow non-linearly for the GPU implementation, transitioning to being slower per iteration than the CPU-based implementation. This is due to the increase in the number of messages being parsed by each agent within the simulation.

3.8 Summary

The work in this chapter provides Contribution C1 of this thesis. A set of models required to evaluate and compare the performance of alternate microscopic road network simulations are defined. The simplified model allows the evaluation of alternate algorithms, data structures and techniques in novel implementations without the burden of implementing the vast set of behaviours and interactions are available within the state of the art commercial simulation packages. Secondly, also as part of Contribution C1, a procedurally generated artificial road network is defined. This scalable network allows the performance of alternate implementations of the simplified model to be evaluated at a range of problem sizes, but also with varying vehicle density by varying other model parameters.

A GPU accelerated implementation of this model is described, using the FLAME GPU framework. The GPU implementation was cross-validated against the reference multi-core CPU simulator, Aimsun 8.1, to ensure that fair comparisons can be made.

Two sets of benchmark experiments were carried out to evaluate the performance of the CPU and GPU simulation implementations as various network properties were scaled. These benchmark results show that the FLAME GPU based simulations can demonstrate comparable performance to the reference CPU simulator, but do not show the significant improvements. The performance impact of communication between individuals within GPU accelerated ABM models is highlighted by these results.

Chapter 4, builds on the work presented in this chapter, by proposing a new communication strategy for use with road network simulations in GPU-based ABMs, and embedded within FLAME GPU.

Figure 3.6: Average iteration execution time for fixed grid of size $N = 16$ against agent population size, averaged over 100 iterations. Results generated using an NVIDIA Tesla K20c GPU.

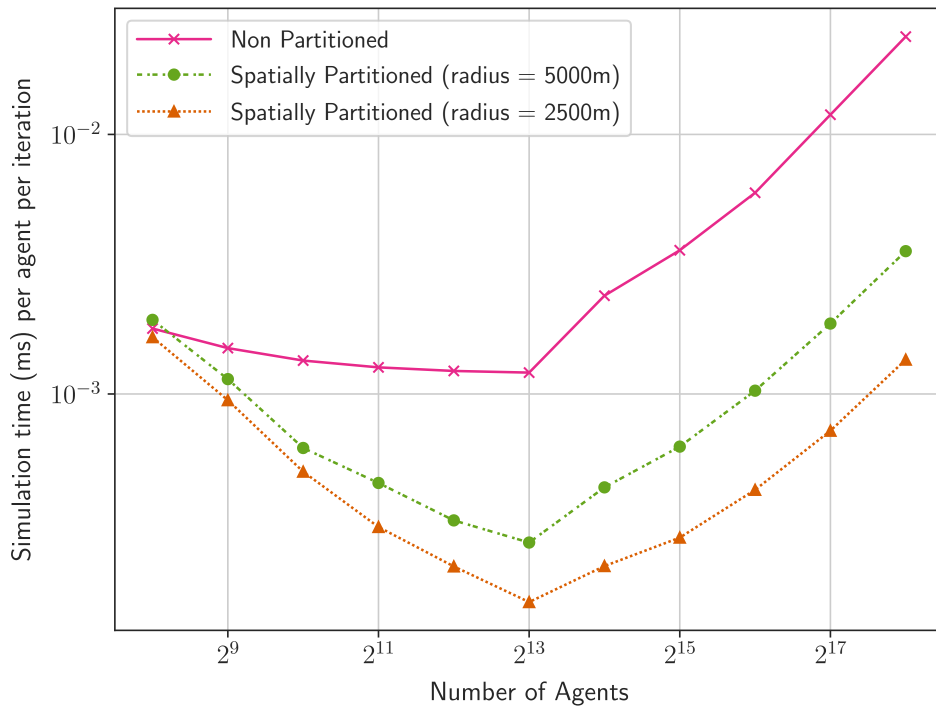


Figure 3.7: Average iteration execution time per agent for fixed grid of size $N = 16$ against agent population size, averaged over 100 iterations. Results generated using an NVIDIA Tesla K20c GPU.

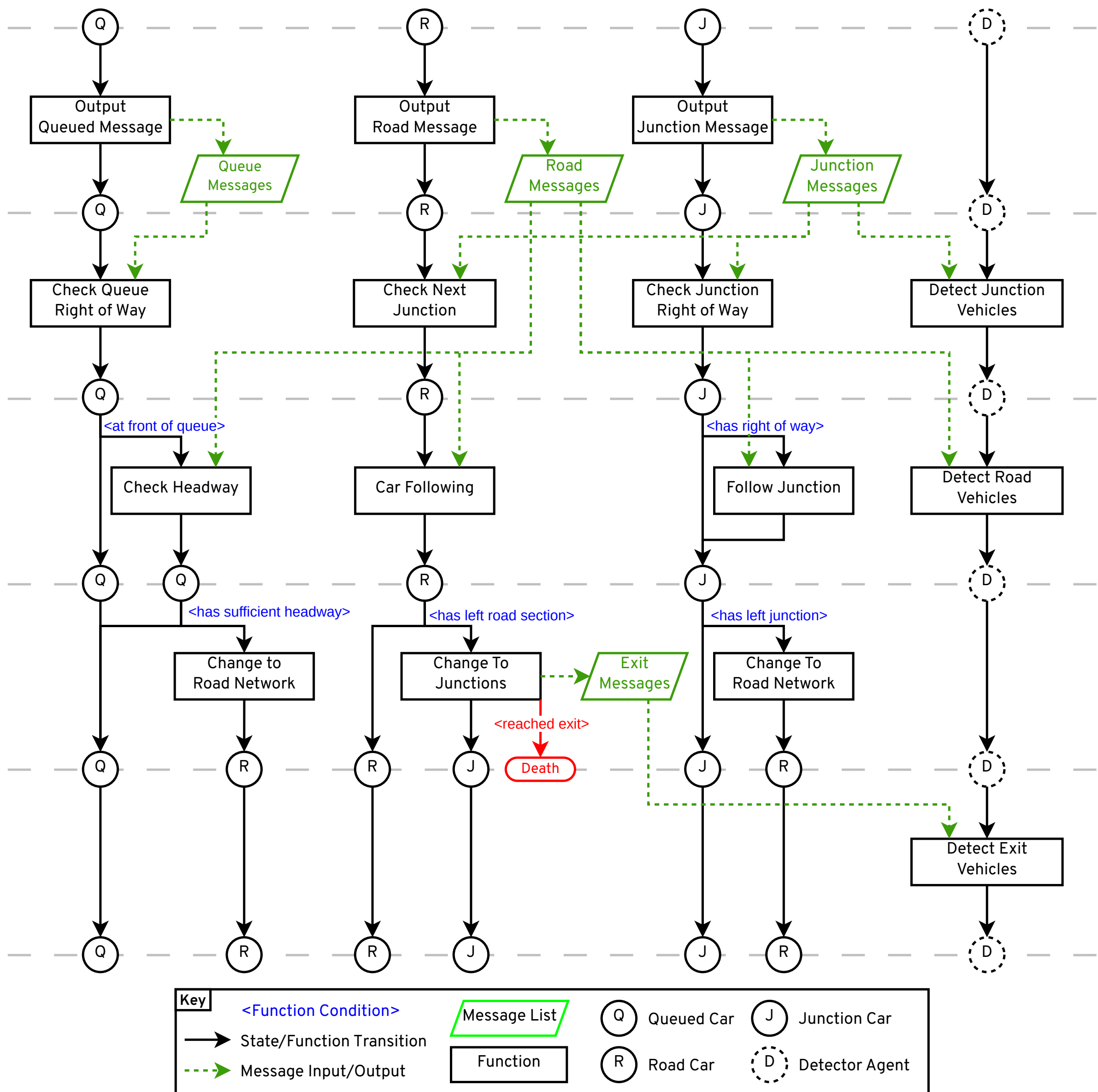


Figure 3.8: FLAME GPU per-iteration state diagram for *vehicle* and *detector* agents. Illustrates the logical control-flow of the agents within a single iteration of the simulation, arranged in simulation layers. Circles represent a type of agent in a given state, as described by the key. Agent functions are shown by rectangles, which can result in a change of state. Message lists are shown in green, with dotted arrows indicating the output and input of messages.

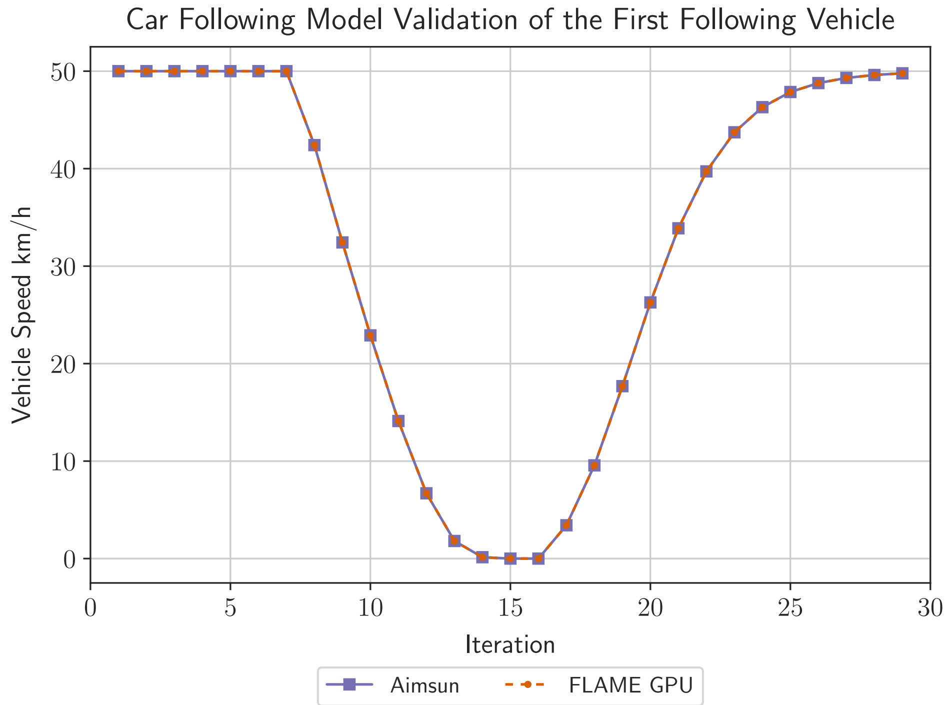


Figure 3.9: Car Following Model Velocity validation. The velocity of the first vehicle in the 100m Car following behaviour validation model with input flow set to 2250 vehicle per hour were manually extracted from Aimsun and FLAME GPU. Comparable results are shown.

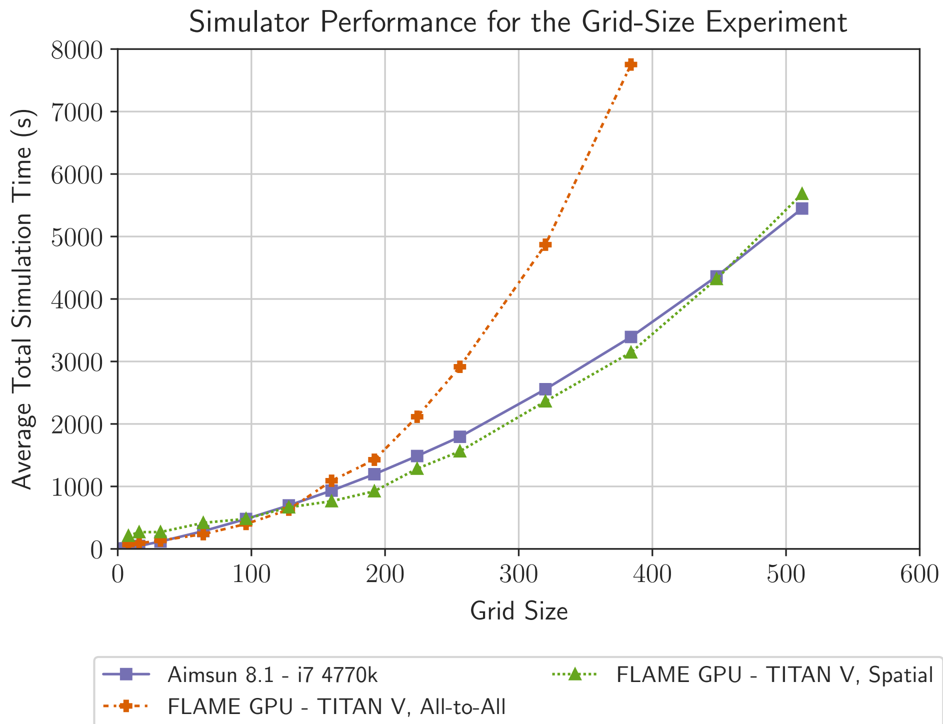


Figure 3.10: Total simulation performance as the total scale of the simulation is increased, for each simulator. Values shown are the average from 3 repetitions. Executed using alternate communication strategies available in FLAME GPU 1.4.

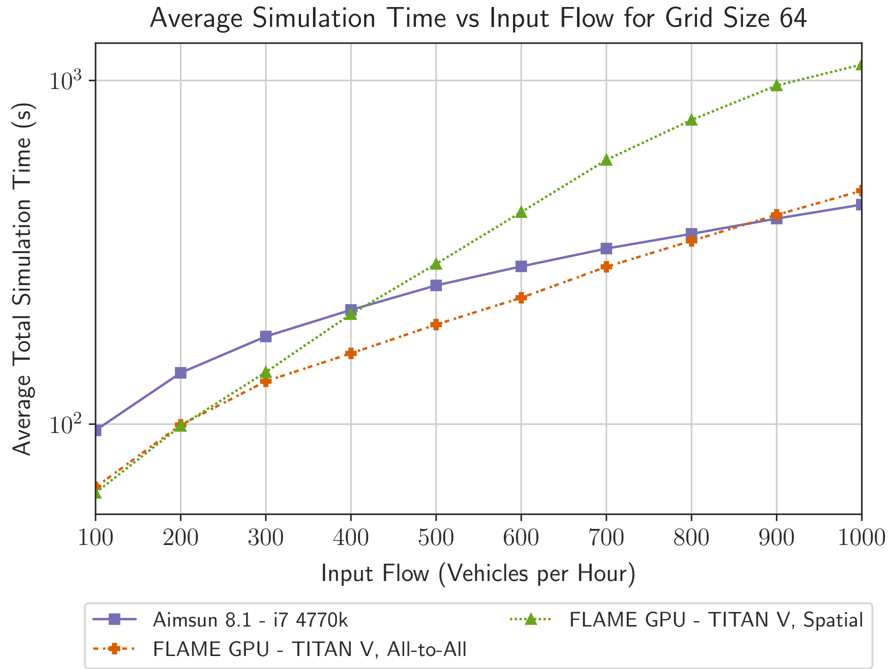


Figure 3.11: Total simulation time against input flow for a procedurally generated road network of grid size 64. Run times shown are the average from 3 repetitions, using a logarithmic scale. The CPU simulation results from Aimsun 8.1 executed on a 4-core, 8-thread Intel i7 4770k are shown in purple. These are compared to results from the GPU simulations using all-to-all and spatially partitioned communication, shown by the orange and green series respectively, executed on an NVIDIA Titan V GPU.

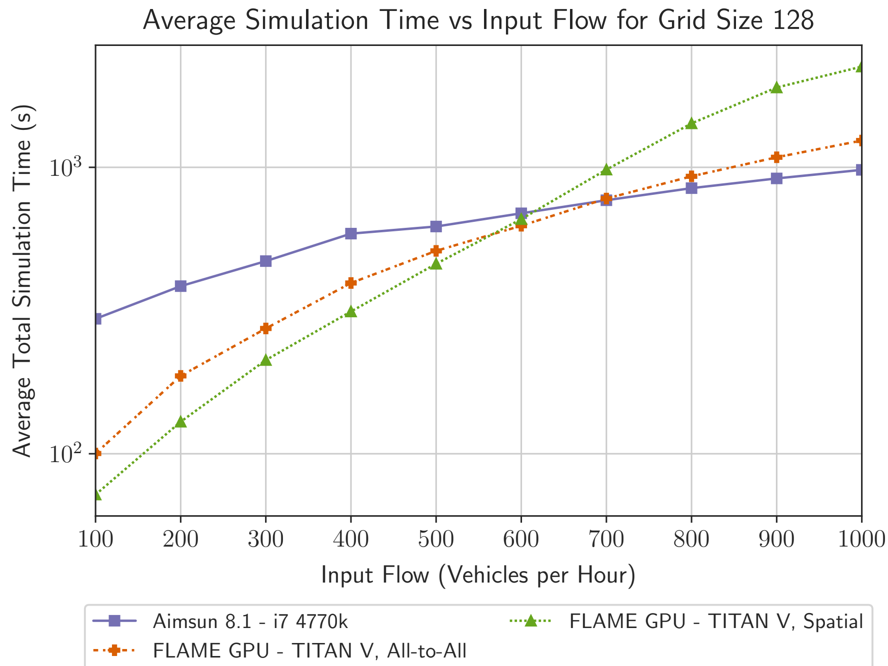


Figure 3.12: Total simulation time against input flow for a procedurally generated road network of grid size 128. Run times shown are the average from 3 repetitions, using a logarithmic scale. The CPU simulation results from Aimsun 8.1 executed on a 4-core, 8-thread Intel i7 4770k are shown in purple. These are compared to results from the GPU simulations using all-to-all and spatially partitioned communication, shown by the orange and green series respectively, executed on an NVIDIA Titan V GPU.

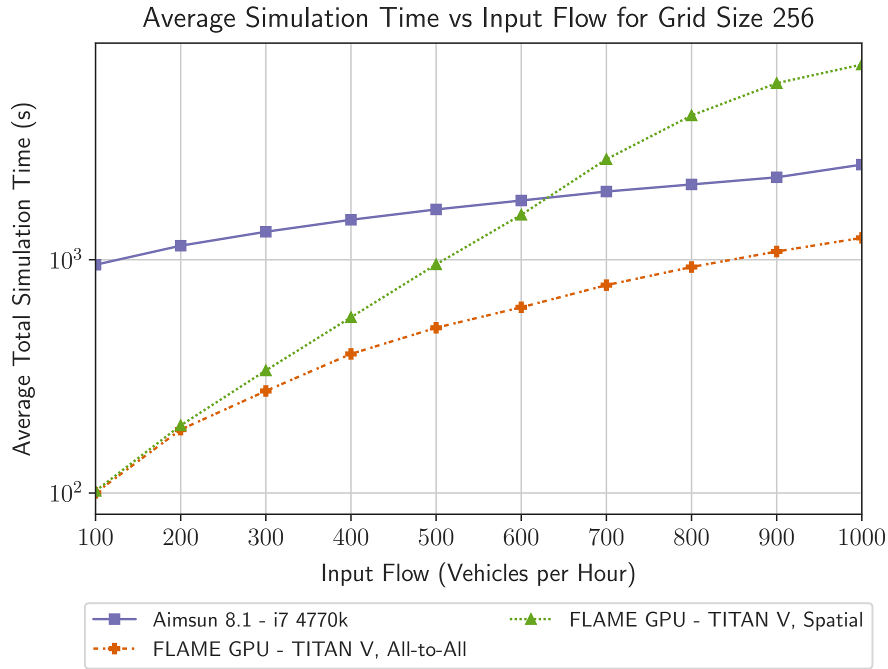


Figure 3.13: Total simulation time against input flow for a procedurally generated road network of grid size 256. Run times shown are the average from 3 repetitions, using a logarithmic scale. The CPU simulation results from Aimsun 8.1 executed on a 4-core, 8-thread Intel i7 4770k are shown in purple. These are compared to results from the GPU simulations using all-to-all and spatially partitioned communication, shown by the orange and green series respectively, executed on an NVIDIA Titan V GPU.

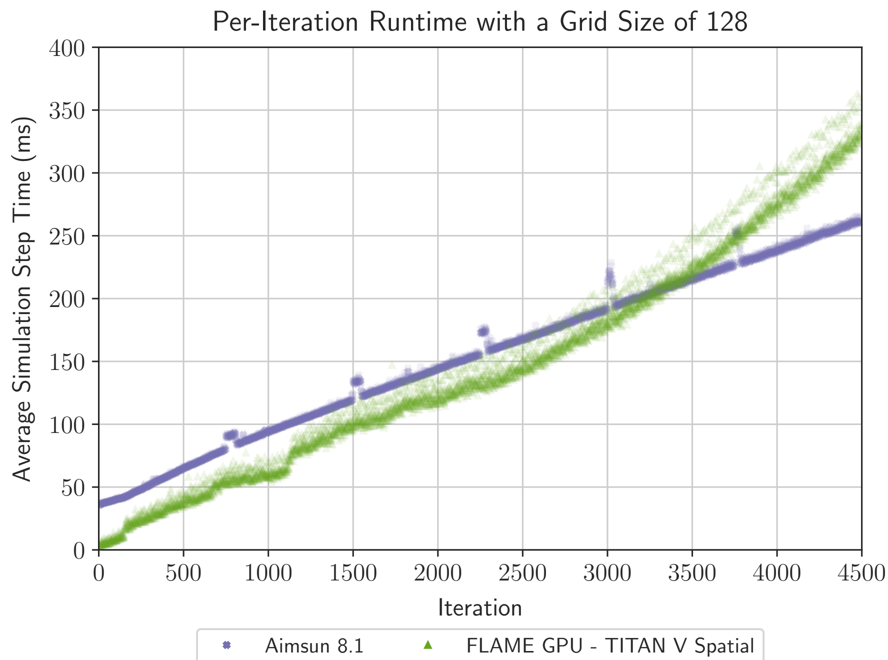


Figure 3.14: Per-Iteration simulation time for a model with grid size 128 and input flow 500 vehicles per hour. Executed using Aimsun 8.1 (purple) on an i7 4770k, and using the FLAME GPU implementation with Spatially Partitioned Communication, on an NVIDIA Titan V GPU.

Chapter 4

Network-Based Communication for Data-Parallel ABM

4.1 Introduction

This chapter builds on the work presented in Chapter 3, which presented a GPU accelerated microscopic road network simulation, and several benchmark experiments which evaluated the performance of the GPU-based implementation against a commercial CPU-based application. The benchmark results highlighted the importance of efficient communication within large-scale parallel microscopic road network simulations.

In this chapter, it is proposed that by coupling the communication data structure and access method to the graph representing the transport network the efficiency of communication can be improved, by reducing the number of candidate messages which must be considered by each agent to gather the necessary information to update the agent state. It is anticipated that this will lead to increased simulation performance. The general purpose graph-based communication pattern, and the implementation of this for many-core processor architectures is described, forming contribution C3 of this thesis.

In order to evaluate the impact of the proposed graph-based communication strategy on Agent Based Models (ABMs) which exhibit similar patterns of communication to road network simulations, an abstract benchmark ABM is proposed. The abstract model is benchmarked at a range of population scales, using three Flexible Large-scale Agent Modelling Environment for Graphics Processing Unit (FLAME GPU) based implementations, using two existing communication strategies and the newly proposed graph-based communication strategy. This model and the associated benchmarking provide Contribution C3.

Finally, the proposed graph-based communication is embedded within the GPU accelerated microscopic road network simulation from chapter 3. This implementation is benchmarked using the same experiments as described in chapter 3, and the performance compared against

the reference CPU results, and previous GPU simulation results. This performance evaluation provides Contribution C4.

The remainder of this chapter is structured as follows. Section 4.2 provides information on message-list based communication within FLAME GPU. Section 4.3 proposes the graph-based communication pattern which will improve the efficiency of communication for road network simulations. An abstract benchmark model is defined in Section 4.4 and used to evaluate the performance impact of the communication strategy. Section 4.5 describes the application of this communication strategy to the simplified Aimsun model from Chapter 3, and evaluates the performance impact through benchmarking. Section 4.6 concludes the chapter.

4.2 Agent Communication in FLAME GPU

Within FLAME GPU, agents use message-lists to communicate with one another, using specialised access patterns to improve the efficiency of communication where appropriate. This employs alternate data structures and message-list traversal algorithms to reduce the size of message lists based on the desired interaction pattern. This specialisation of communication can have significant impact on simulation performance, especially for large-scale models where global communication can account for the majority of runtime.

Existing techniques for message-based communication available within FLAME GPU 1.4 allow for global communication (all-to-all) or local communication based on spatial locality (discrete partitioning and spatial partitioning) [179].

All-to-all messaging provides indirect global communication between agents. Discrete messaging restricts communication to local neighbourhoods in 2D discrete space, while spatially partitioned messaging restricts the communication to the local region surrounding the agent in 2D or 3D space. These communication patterns are not ideal for some of the important models used in road network microsimulation, resulting in inefficient communication, as previously shown in section 3.7.5. In road network models, typically communication between agents is restricted by the road network itself. Road network models such as car following or lane changing models require information from neighbouring vehicles on the same or directly connected road section. Other models such as gap acceptance or give-way modelling at junctions will typically require information from sections of road related approaching or leaving the junction in question.

In these cases, neither all-to-all communication or fixed-radius spatial partitioning are optimal for large-scale simulations. Global communication is unsuitable as it is highly inefficient, while fixed-radius spatial approaches do not consider the complex topology and speeds of the road network. Both approaches result in large message lists to ensure that all required information would be captured, and therefore a large portion of runtime is spent iterating these

messages.

Limiting the communication to spatial partitioning does improve performance compared to global communication (as shown in Section 3.7.2 and [29]), but still yields overly-large message lists considering urban environments which are densely packed with disconnected sections of road, and also when high speed sections of road are included, resulting in larger-communication radii being required.

4.3 Graph Based Communication

Ideally communication should only need to occur within the constraints of the network, which is typically represented using a graph or series of graphs. For the purposes of the benchmark road network simulation model described in chapter 3, which consists of single lane sections of roads and stop-sign-based, yellow-box junctions, communication is limited to the current road network edge, the next road network edge, or the current junction (graph vertex). Existing techniques for graph-based communication exist within parallel and distributed agent based frameworks such as D-Mason and Repast HPC, however, these strategies use individual agents as the vertices of the graph with the graph edges representing the relationships between agents [20], [21].

The proposed graph based communication method works as follows. Individual agents are coupled with the graph, maintaining their location within the graph at all times. For a network-based communication strategy, the messages are also coupled to the underlying graph. When an agent outputs a message, it must include the edge or vertex for which the message is coupled. The message list of all messages can then be sorted by the edge or vertex to align memory and a sparse data structure is constructed to enable access to relevant messages of the data structure. The data-structure construction is similar to that used for fixed radius communication [181] commonly applied to many particle-based GPU simulations [246], which identifies the position within the message list of the first message for each edge or vertex in the network, and the number of messages for each edge or vertex. The sorting of the message list and construction of the message delimiting data structure can be performed efficiently using a *counting sort*. High performance implementation of counting sort is reliant upon GPU fundamentals like cache-aware atomic operations. As the number of edges or vertices within the network are known, the atomic counting sort constructs a histogram containing the number of messages for each edge or vertex within the network. The previous value of the per-element counter is stored for each message. An exclusive scan can then be performed over the histogram of message counts to find the starting index for each bin of messages once sorted. Once the starting position within the global list is known, combined with the position of each message within the local bin, message data can be sorted within global device memory in parallel. The histogram of

message counts and the starting position of messages for each graph element are then used to access messages within the data structure, given a graph element index to access. If an alternate sort algorithm were used, such as a radix sort, then the number of messages stored in each bin, and the offset to the start of each bin would also need to be calculated, leading to additional costs. The network-based communication strategy is general enough to be applied to any GPU multi-agent simulation, where communication along a graph-based data structure is required, such as social network modelling. Figure 4.1 visually demonstrates an example, showing the overall process.

When messages are requested by agent functions, the partition boundary matrix is queried to find the index of the first sorted message for the requested graph element. Individual threads then load the relevant data into local memory within the Streaming Multiprocessor (SM) to provide access to the data as if it were stored within a struct, rather than the Structure of Arrays (SoA) data structure used to store data within global device memory.

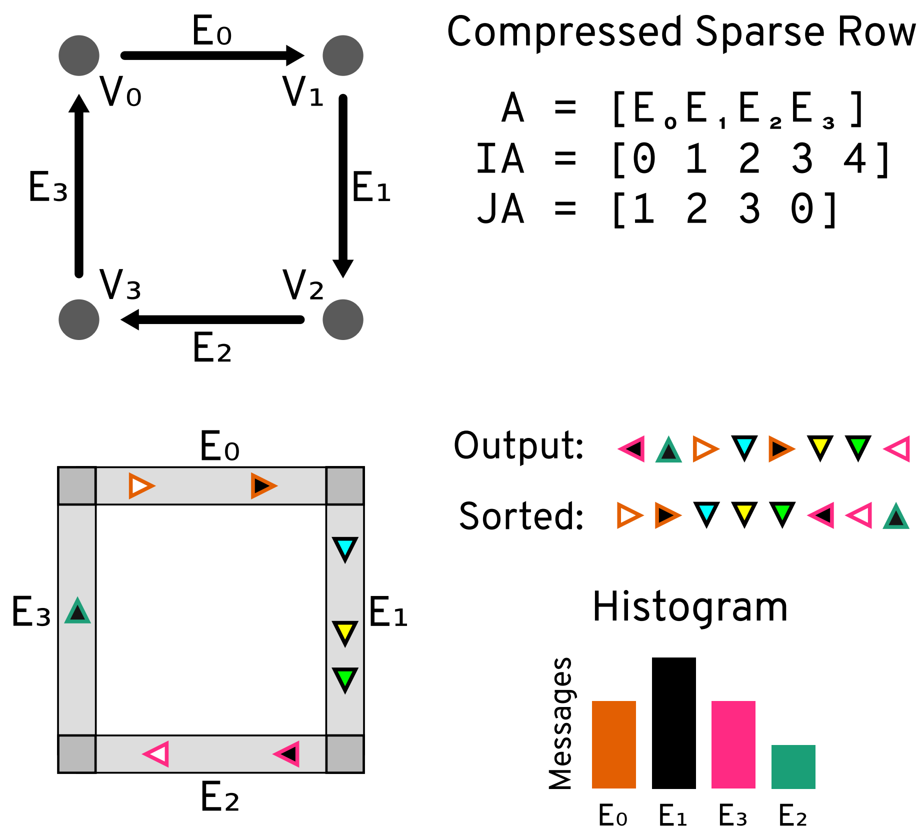


Figure 4.1: An example visually representing the data structure construction for the message processing algorithm. A simple directed graph is shown, containing 4 edges and 4 vertices, coupled with the equivalent CSR representation of the graph. 8 example agents represented by coloured triangles are shown on the road network represented by this figure, and the order of messages as output by these agents. The sorted order is then shown, coupled with the histogram, which provides access to the appropriate set of messages for a given target edge.

The graph-based communication strategy was embedded within FLAME GPU as a part of

the `1.5.0` release[1], by commit `550f042`[2]. To make use of the new communication strategy, several key changes must be made to the FLAME GPU implementation. The XML Model File must include a definition of the graph, so that properties can be accessed on the GPU. This includes the data types for vertex and edge indices, plus additional properties which can be associated with vertices or edges, such as the position of a vertex in space, or the weight of an edge. Message lists which wish to be accessed through the graph-based communication strategy must be defined within the model file as using the `gpu:partitioningGraphEdge` message partitioning strategy. This definition includes declaring which message variable which must contain the index of the graph element the message is associated with. Graphs are loaded from disk at the start of a simulation, the CSR representation is constructed and moved onto the GPU. Message output from within an agent function behaves the same as any other partitioning scheme within FLAME GPU. When accessing messages from the message list, the modeller must provide the graph element index for which they wish to access messages from as an additional parameter to the `get_first_x_message` method. The message list can be queried by multiple times within an agent function, to access messages for separate graph elements if required. A selection of methods to access the graph data structure are provided to support this.

4.4 Abstract Graph Communication Benchmark Model

To evaluate the communication strategy independently of a complex road network model implementation, and aid implementation within the FLAME GPU framework, a simple test model was designed. By designing a very simple model with light-weight agents with low memory requirements, the effects of message list specialisation can be emphasised and more clearly measured and identified with profiling tools. This is especially beneficial when considering memory bandwidth, which is often the limiting factor in FLAME GPU agent functions which iterate message lists.

The abstract model is inspired by the flow of fluid through a series of connected of pipes. Agents represent individual units of fluid, and the network of pipes which makes up the environment is represented by a graph, where each edge can be thought of as a section of pipe with a fixed upper limit on capacity. Individual agents traverse these pipes according to simple properties and rules, but do not block each other within a pipe section. The only source of conflict to be resolved is when traversing to the next section of pipe.

The network of *pipes* consists of vertices with `id`, `x`, `y` and `z` variables, encoding the location of the vertex in space for visualisation purposes. Edges in the network have an `id`, `source` and `destination` which encode how the edge is connected to vertices, along with `length` and

[1]https://github.com/FLAMEGPU/FLAMEGPU/releases/tag/v1.5.0
[2]https://github.com/FLAMEGPU/FLAMEGPU/commit/550f04209b8ad6c8eb91e57c8e2acebd171a81b8

capacity defining how long the edge is and the number of agents which can exist within the edge at a given point in time.

A single agent type is used in this model. The agent has a unique id, a position within the network (the currentEdge and position along the edge) and the edge which it is currently travelling towards nextEdge. Additional agent variables are used for visualisation (x, y, z & colour), aggregate statistics (distanceTravelled & blockedIterationCount) and memory values which persist between agent functions (nextEdgeRemainingCapacity & hasIntent).

The model has four parameters: the seed for random number generation (SEED), the initial population of agents (INIT_POPULATION) and parameters controlling the minimum and maximum speeds which agents will be initialised with (MIN_SPEED & MAX_SPEED).

Agents are initialised in a FLAME GPU initialisation function, executed in serial on the host. Agents are distributed uniformly on the edges of the graph, subject to any capacity constraints. The position of the agent along the edge is randomly sampled from a uniform distribution. The nextEdge is selected randomly from the connected edges. Agent speeds are randomly sampled from a uniform distribution, and between the parameterised minimum and maximum speeds. Other variables are initialised to sensible default values (i.e. 0), or calculated based on the network (x, y, z).

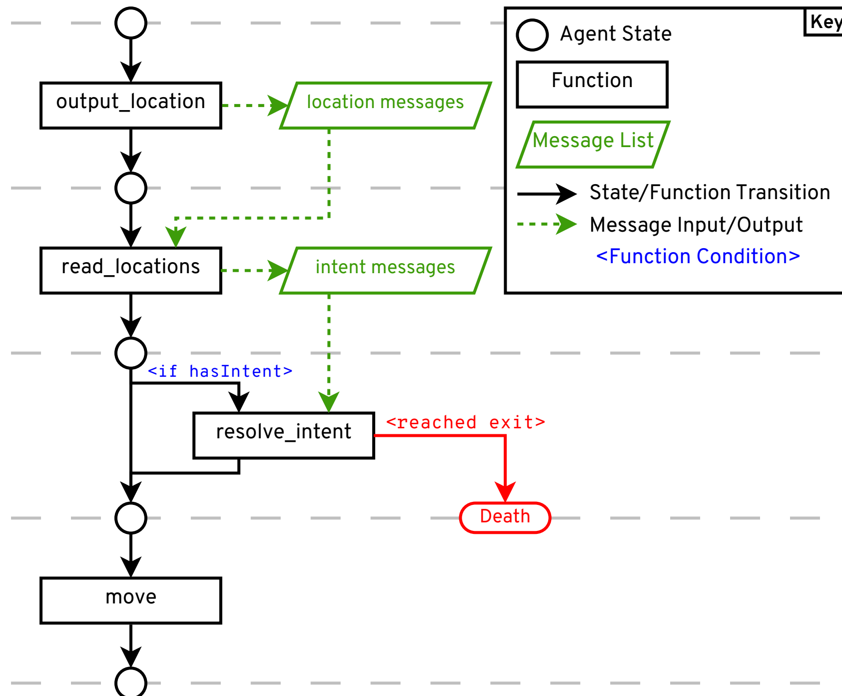


Figure 4.2: The FLAME GPU state diagram for the restricted flow graph model. It shows control flow within a single iteration of the simulation for the agents which only have a single state. Agent functions are represented by black rectangles, with green parallelograms representing message lists. The state diagram shows the 4 agent functions, arranged in layers which are executed sequentially.

Figure 4.2 shows the FLAME GPU agent state diagram for each iteration of the simulation using circles to show agent states, rectangles for agent functions with solid arrows for transitions

between to show the flow of agent functions and states. Message lists are represented by green parallelograms with dotted arrows used to indicate the output and input of message lists by agent functions. In the first iteration layer, agents execute their `output_location` behavioural function, where the publicly visible information is broadcast into the `location` message list, and reset variables used on a per-iteration basis. Agents then execute the `read_locations` function in the second layer. In this function agents iterate the `location` messages to find the number of individuals on their target `nextEdge` to determine if there is capacity to make the transition or not, if the agent is close enough to the end of their current edge and moving with sufficient speed to make the transition. If there is sufficient capacity, and the edge transition is required the agent broadcasts an optional message to the `intent` message list, declaring that they wish to move to the target edge.

If the agent does need to transition to the next edge, it executes the `resolve_intent` function in the third layer. In this function Agents iterate the intent messages to find how many agents wish to move to the next edge, and the position within the queue to move to the next edge, based on the unique id of the agent. Combining this with number of agents on the next edge (from the previous agent function) the agent determines if it can move to the next edge or not. If the agent can move to the next edge, it makes the transition and randomly selects a new edge from the connected edges at the destination vertex. The agents position is set to a negative value, indicating its location at the start of the iteration. If the agent has reached a terminating edge, where there are no connected edges at the next vertex, it will leave the simulation (death).

Finally, in the fourth function layer agents execute the `move` function. Agents simply move according to their speed and position within the current edge. If they are blocked from traversing to the next edge, i.e. they were unable to move to the next edge due to a capacity constraint, this is recorded by the agent to be aggregated once the simulation is completed. Once the expected number of simulation steps has completed, a FLAME GPU *exit function* is executed, which performs reductions of per-agent properties into aggregate values which are used as simulation outputs.

This model does have several limitations worth noting. As agents all execute concurrently, some agents may not transition to the next edge even if new capacity becomes available due to agents exiting the edge at the same time. This could be resolved with a much more complex conflict resolution method and lookahead, but this would greatly increase complexity and serialisation within the model. Instead, by using a sufficiently short time-step (or low speed values) this effect is minimised. It is also possible to select parameters and networks which when combined result in a non-viable simulation, although this can typically be detected at initialisation time and the simulation can be gracefully aborted.

4.4.1 Benchmark Results

This proposed model allows the performance impact of the communication strategy to be benchmarked independently of the more complex agents and agent behaviours than in a road network microsimulation model. This benchmark experiment evaluates the impact of the communication strategy as the density of agents is increased for a fixed size environment, for behavioural models which require a similar communication strategy to a road network simulation.

The network of "*pipes*" used for this set of benchmarks is a very simple 2D grid. Figure 4.3 shows this layout with 4 rows and 4 columns. Graph vertices are arranged in a 2D grid, with edges connecting to each immediate neighbour in the four cardinal directions. The graph used is directed, containing 2 edges between a given pair of vertices in opposite directions. For example, the edges `A -> B` and `B -> A`.

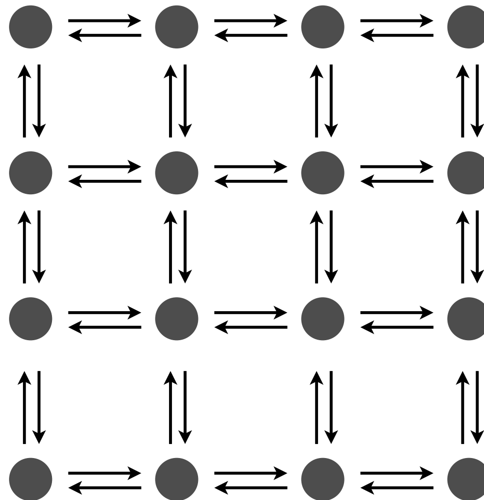


Figure 4.3: A 4 row, 4 column example of the 2D grid of "pipes" used for the abstract graph communication benchmark model. Each vertex (grey circle) is connected to the neighbouring vertices in each cardinal direction, with a separate edge (arrow) in each direction, as properties such as capacity could be different in each direction. The benchmark experiments used a 32 by 32 grid.

Benchmarks were carried out on a fixed size network, containing 1024 vertices arranged in a 32 by 32 grid and 3966 edges, each with a capacity of 256 and a length of 50. The agent population is scaled from 2^{10} to 2^{19} (1024 to 524288). Each simulation completed 1000 iterations and was repeated 3 times to find the mean runtime. Figure 4.4 shows the average runtime for this benchmark using each communication strategy executed on an NVIDIA Titan V GPU. For graph partitioning the smallest simulation completed in 0.408s and largest in 9.062s on average. Spatial partitioning completed the smallest simulation in 0.640s and largest in 1243.376s. All to all communication took 0.650s for the smallest simulation, and 1179.514s for the largest simulations.

These results clearly demonstrate a significant reduction in simulation runtime based on a change in communication pattern, showing performance improvements of up to 137x compared

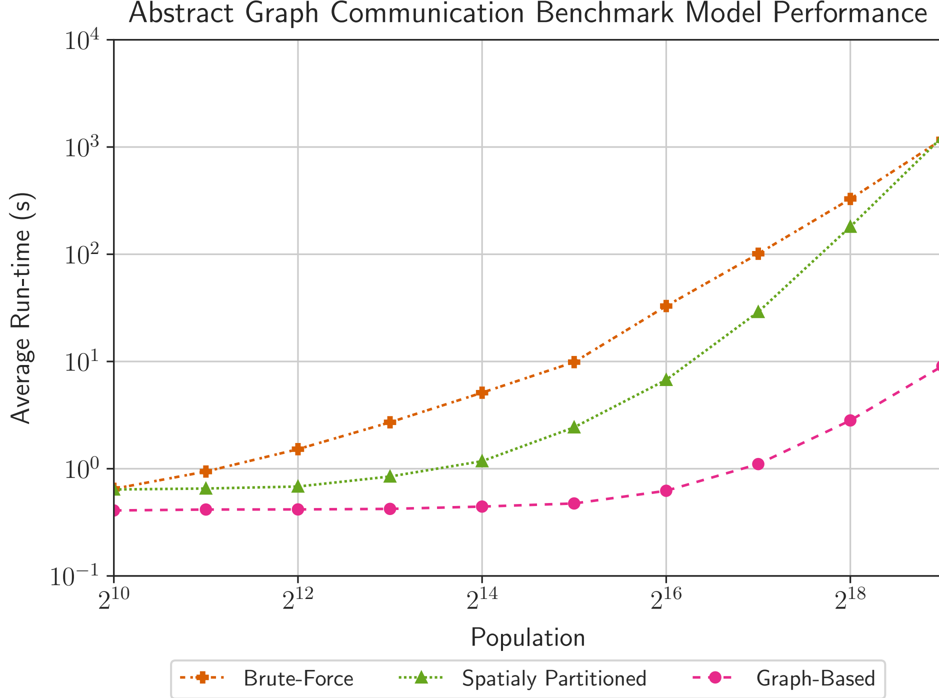


Figure 4.4: Benchmark results for the Restricted Flow Graph model, for the two existing FLAME GPU communication strategies (Brute-force and Spatially Partitioned) and the proposed graph-based communication strategies, on a fixed size graph with varied populations. Results were collected using a NVIDIA Titan V GPU. The average of 3 runs is presented in seconds. This benchmark model has only been implemented for Graphics Processing Units (GPUs).

to spatially partitioned messaging, and up to 130x compared to all to all communication. Improvements are observed across all population scales, with the most significant improvements for larger scale simulations, where the scale of message lists is much larger for spatially partitioned or non-partitioned messaging, due to the greater number of agents and increased agent density on the fixed size network.

At the smallest scale of simulation, all-to-all and spatially partitioned communication show very similar runtimes, although already slower than graph based communication. As the population increases, all-to-all communications results in steadily increasing run times as message lists grow directly with population. Spatially partitioned messaging initially shows a much slower rate of runtime increase with population increase, however, as agent density increases the message lists grow more significantly. Once populations are very large and density is very high, although message lists are smaller than for all-to-all communication, the overhead costs (such as data movement, context creation and kernel launch overheads) of this approach outweigh the benefits, with spatially partitioned messaging and all-to-all communication once again taking approximately the same length of time to complete. Graph partitioning on the other hand shows almost no change in runtime initially as the population grows, as message lists are still very small. Once density is sufficiently high, and the average number of agents on an edge becomes larger the message list size once again results in a degradation of performance, but

with a slower rate than the alternate approaches.

4.5 Application to Simplified Aimsun Model

Once the proposed graph communication strategy was implemented and shown to improve communication efficiency and reduce simulation runtime in the communication benchmark model defined in Section 4.4, it was applied to the microscopic road network model described in section 3.3 in order to understand the impact of this message list specialisation on transport network simulation.

Figure 4.5 illustrates how the change in communication strategy can have a significant impact on the size of message lists in vehicle modelling, using a relatively small example for simplicity. The figure shows a portion of the grid-based transport network, with grey rectangles representing sections of road and coloured triangles representing individual agents. The colour of the triangle shows the messages processed by a single agent, shown in white in cell 5, for each message partitioning technique. The communication radius for spatially partitioned messaging must be at least as large as the longest road section to ensure all appropriate messages are found. Using the new graph-based partitioning the agent only processes 5 messages (indicated by orange borders), compared to 18 messages for spatially partitioned communication (highlighted in blue), and 42 messages using all-to-all messaging. This reduction in the size of message lists iterated by each agent leads to considerable performance improvements for this type of model. The scale of reduction in message list size is even more significant in larger models, with larger, denser populations of agents.

As the communication strategy was integrated into FLAME GPU, changes to the model implementation were minimal. `XMLModelFile.xml` was modified to specify `<partitioningOngraph>` rather than `<partitoningNone/>`, with the appropriate parameters for each message list. The message iteration loops within `functions.c` were modified to use the appropriate, dynamically generated FLAME GPU API functions, and `if` statements used to check the edge of the message matched the target edge were removed, as this is now handled by the communication strategy.

The set of benchmarks previously described in section 3.5, the grid-scale experiment (section 3.5.1) and input-flow experiment (section 3.5.2), were repeated using the modified implementation using the new communication strategy, and performance compared to previous results for both Aimsun 8.1 on the CPU, and the FLAME GPU simulations using GPUs sections 3.6 and 3.7.5.

4.5.1 Grid-Scale Experiment Results

As in Section 3.5.1, the performance impact of the total problem scale is evaluated by varying the grid-scale of the procedurally generated network, using the same parameters previously

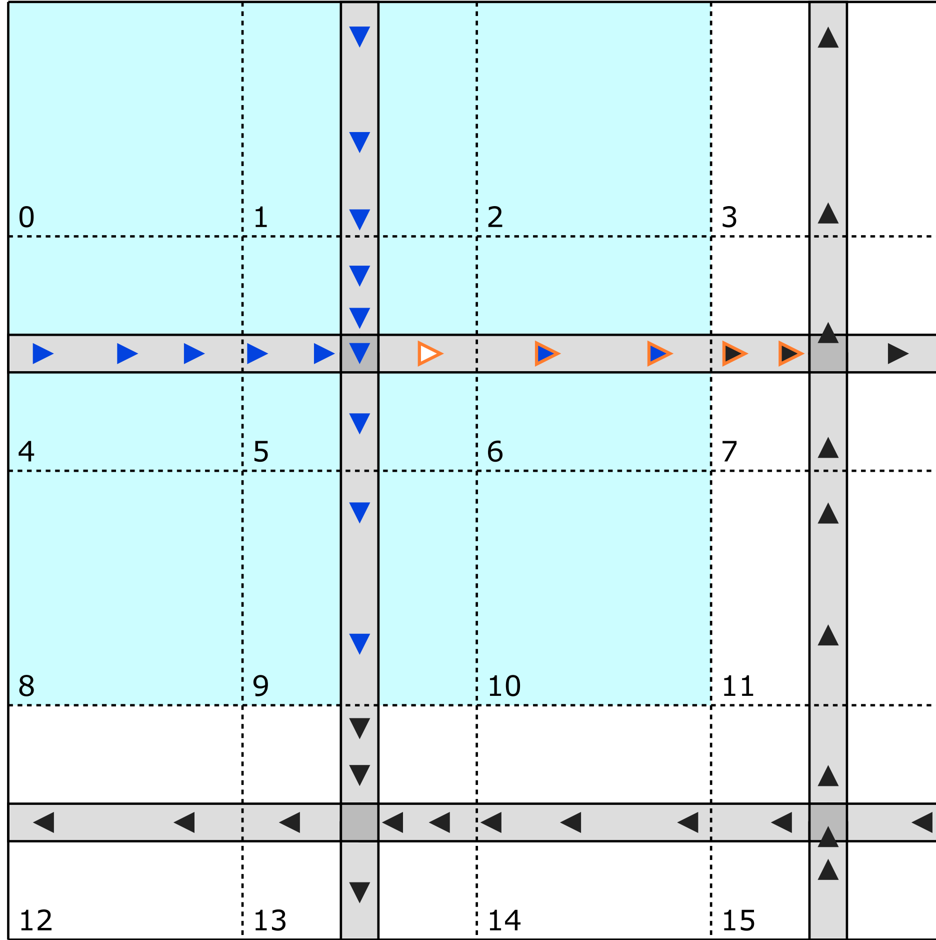


Figure 4.5: The effects of alternate FLAME GPU communication strategies on car following models is shown for a section of a grid-based road network, for a single agent represented in white. All-to-all communication results in 42 messages being parsed by the single agent. Spatially partitioned messaging results in the 18 messages from the agents in the blue shaded region, while the graph-based communication strategy results in only 5 messages from the orange bordered agents being processed. The spatially partitioned radius used for this illustration would be insufficient for accurate modelling, and is only used for illustrative purposes.

described in table 3.3. Figure 4.6 show the average simulation time for 3 repetitions of the simulation for the CPU simulations and for three communication strategies using a NVIDIA Titan V GPU. Additional benchmarks were performed using NVIDIA Titan Xp and NVIDIA Titan RTX GPUs to provide insight into the performance difference between different generations of GPU. The figure shows that the graph-based communication strategy outperforms the CPU-based simulator at all but the smallest scale, where the overhead costs associated with using GPUs outweighs any reduction in runtime due to a lack of parallelism and utilisation of the hardware.

At small scales, the CPU simulator outperforms all GPU implementations, regardless of the communication pattern used. The all-to-all GPU simulator shows a minor performance improvement to the CPU simulator for grid sizes between 64 and 160, and the spatially partitioning simulator shows slightly improved performance between 128 and 448, as shown in the

previous chapter. Graph communication shows much greater improvements in performance, compared to both the CPU and other GPU simulators. For grid sizes of 32 and larger at-least a factor of 2 performance improvement is shown, growing to 59.5x for a grid size of 512. This corresponds to a real-time-ratio (RTR) of 36 for a simulation containing over 500,000 vehicles.

The graph communication technique exhibits considerable performance improvements compared to both all-to-all and spatially partitioned messaging. Compared to brute-force messaging, improvements of between 1.04x and 104.6x are shown, while improvements of between 3.2x and 62.0x are demonstrated compared to spatially partitioned messaging with a sufficiently large communication radius to ensure correct results.

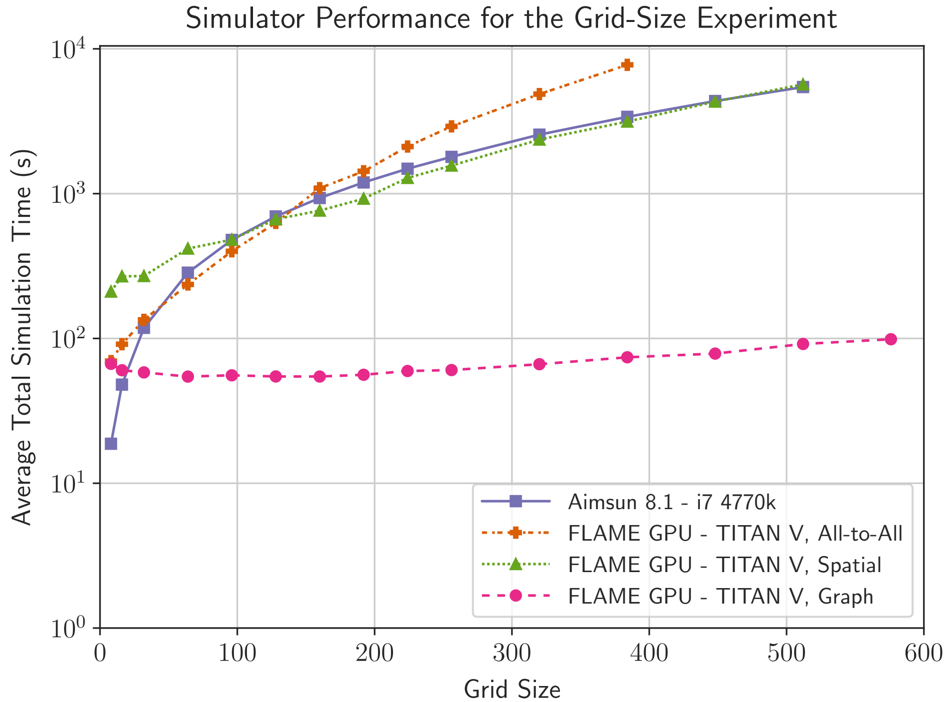


Figure 4.6: Total simulation performance as the total scale of the simulation is increased, showing the performance of the CPU simulations, and the alternate GPU simulations using a single GPU. Values shown are the average from 3 repetitions. A logarithmic scale is used to improve visibility of similar total simulation times.

The Grid-size experiments were executed on multiple generations of NVIDIA GPU to see how the generation and scale of the GPU effects performance compared to one another, and the reference CPU results. Figure 4.7 shows the results for the grid-scale experiments using the CPU and using the graph-based communication strategy on three generations of NVIDIA GPU: Pascal (Titan Xp), Volta (Titan V) and Turing (Titan RTX). Once again, this shows that regardless of the GPU the CPU simulator offers the best performance for small-scale simulations, but GPUs are much more suitable for larger-scale simulations, offering significant performance improvements, of up to 67x, corresponding to a RTR of 44.7 for a simulation containing over $500,000$ vehicles. The different GPUs show similar performance curves. The Pascal (Titan Xp)

and Volta (Titan V) architecture GPUs show very similar initial performance, but as the scale of simulation grows the performance difference increases, up to a factor of 1.8 in favour of the Titan V GPU. This can mainly be attributed to the larger number of processor cores, and significant increase in memory bandwidth. The Turing architecture GPU, the Titan RTX, initially shows much higher levels of performance, of over a factor of 2 compared to the Titan V and Titan Xp GPUs. However, the Titan RTX does not scale as well with simulation scale compared to the Volta GPU, resulting in similar performance for the largest simulations. This scaling behaviour is most likely due to the higher number of streaming multiprocessors present and therefore a larger number of resident threads in the Titan V compared to the Titan RTX, resulting in reduced serialisation once the device is over-subscribed. Memory bandwidth is comparable between both devices. The Volta-architecture Titan V also contains a much greater number of FP64 cores compared to the Pascal and Turing GPUs, however, the GPU implementation uses FP32 floating point values so this will not impact performance.

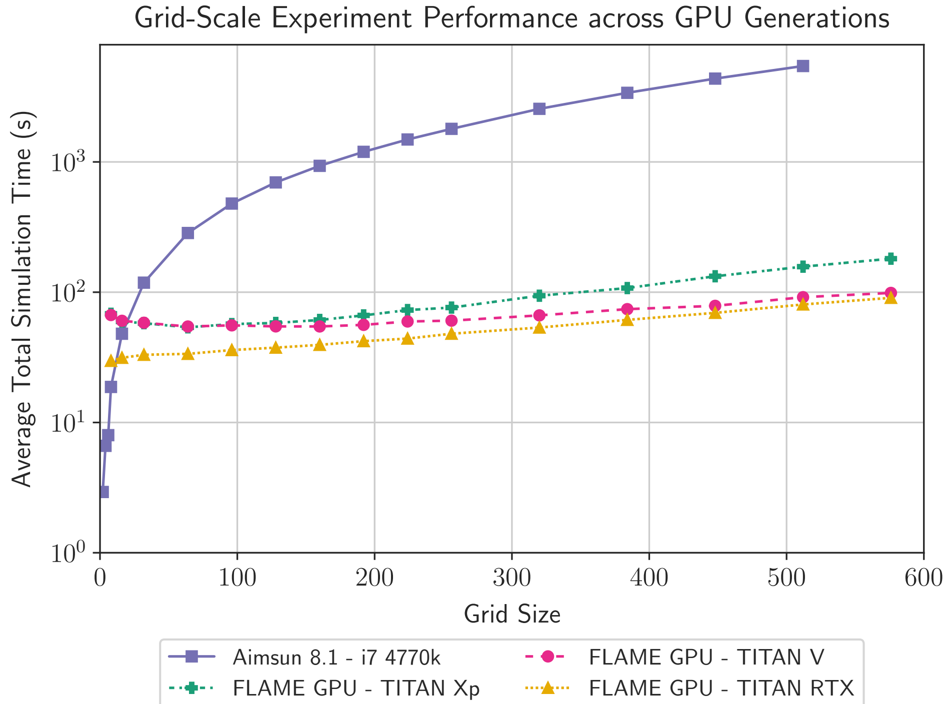


Figure 4.7: Total simulation performance as the total scale of the simulation is increased, for each simulator. Values shown are the average from 3 repetitions, for the CPU simulator and for the graph-based messaging GPU simulator using multiple GPUs. A logarithmic scale is used to improve visibility of similar total simulation times.

4.5.2 Input-Flow Experiment Results

The input-flow experiments from Section 3.5.2, which are used to evaluate the performance characteristics as the agent-density is increased for fixed size environments, were repeated using the new communication strategy. The parameters from table 3.4 were used once again, executed

using an NVIDIA Titan V GPU (5120 core), and using a single thread on an Intel Core i7-6850K for Central Processing Unit (CPU)-based aspects of the simulation.

Figures 4.8 to 4.10 show the average simulation time for 3 repetitions of each simulation. These figures expand upon the input flow experiment results for the CPU, all-to-all GPU and spatially partitioned GPU results from Section 3.5.2, with the addition of the graph-based communication simulation times shown by the dashed pink series. All three figures show that the graph-based messaging strategy shows a significant reduction in simulation time compared to the reference CPU results, and to the alternate communication patterns used. The relationship between the input flow per edge and total simulation time is also improved, with a shallower gradient maintaining the performance advantage of the GPU implementation as the density of the problem increases. The fixed-size network simulations with a grid size of 64, the GPU accelerated simulations using graph based communication shows average speed-ups of between 2.2x and 6.7x compared to Aimsun 8.1. Graph communication speed-ups of between 8.0x and 13.9x are shown for a grid size of 128, and between 25.0x and 31.6x for a grid size of 256.

Comparing the communication strategies, graph-based communication shows speed-ups of between 1.5x and 12.7x compared to all-to-all partitioned across the three grid-scales, and between 1.5x and 65.3x compared to all-to-all partitioning across all 3 selected grid sizes on an NVIDIA Titan V GPU. This demonstrates that the communication strategy has a significant impact on the overall performance of the simulations, especially for larger-scale simulations.

4.5.2.1 Per-Iteration Runtime

The per-iteration run-time of the simulations can provide further insight into the relative performance difference between simulation implementations, as previously examined in Section 4.5.2.1. Figure 4.11 shows this for simulations with a grid-size of 128. CPU and spatially partitioned GPU results are repeated from figure figure 3.14, with additional results using the graph-based communication strategy executed using an NVIDIA Titan V GPU. The new graph-based GPU results (pink) show the greatly reduced per-iteration runtime compared to both the Aimsun 8.1 and FLAME GPU results, with a slow rate of increase in per iteration runtime as the simulation progresses and vehicle population increases. The per-iteration runtime is initially comparable to the other GPU results. As the simulation progresses and the population within the simulation grows, the previously dominant cost of message iteration is reduced due to the more efficient access to the potentially relevant messages. The upper bound of population for this grid-scale and input-flow combination is $128,000$ agents, as such the effects of over-subscription of the GPU are not seen in this figure, as the Titan V GPU supports up to 168,000 resident threads. If the device were over-subscribed there would be a change in gradient once over-subscription is achieved.

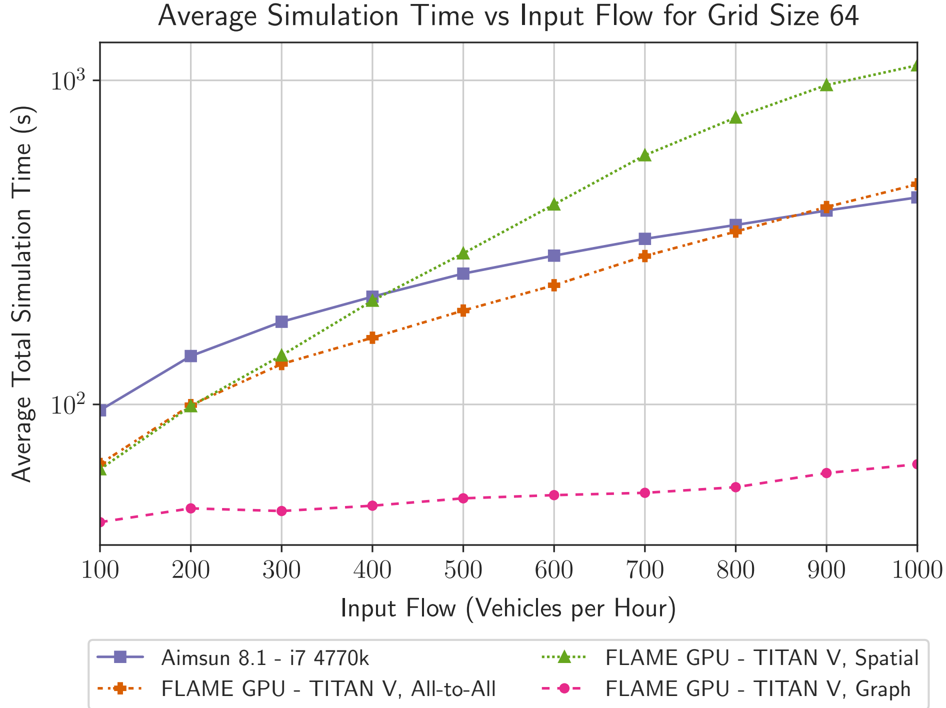


Figure 4.8: Total simulation time against input flow for a procedurally generated road network of grid size 64. Run times shown are the average from 3 repetitions. A logarithmic scale is used for the y axis. The results shown include the reference CPU simulator, Aimsun 8.1, executed on an i7-4770k (purple); and the results from FLAME GPU executed on an NVIDIA Titan V GPU using all-to-all (orange), spatial (green) and graph-based (pink) communication strategies.

4.5.2.2 Kernel-Level Benchmarking

Further insight into the effectiveness of this communication pattern can be gained by viewing the performance of kernels executed on the GPU which make up an individual agent behaviour. By looking at the individual steps, rather than the total time of the simulation or the time of a specific behaviour, the relative effects on message output and input can be evaluated for each communication technique. In this case, the agent functions involved in the agent following model for vehicles on sections of road were profiled to capture performance of the relevant kernels.

The average time taken by the agent function which involves message output is shown in Figure 4.12a, including the time required to construct the message list data structures. This demonstrates that all-to-all communication has the lowest overhead cost for message output, whilst the new graph-based communication strategy has the highest overhead cost. The higher overhead costs of the graph-based communication strategy is expected, as the data structure required to efficiently access graph-based messages is finer-grained than required for the spatially partitioned approach using an appropriate radius. This leads to a longer running scan operation due to the greater the number of elements.

Figure 4.12b shows the average runtime of the GPU kernel which implements Gipps' car

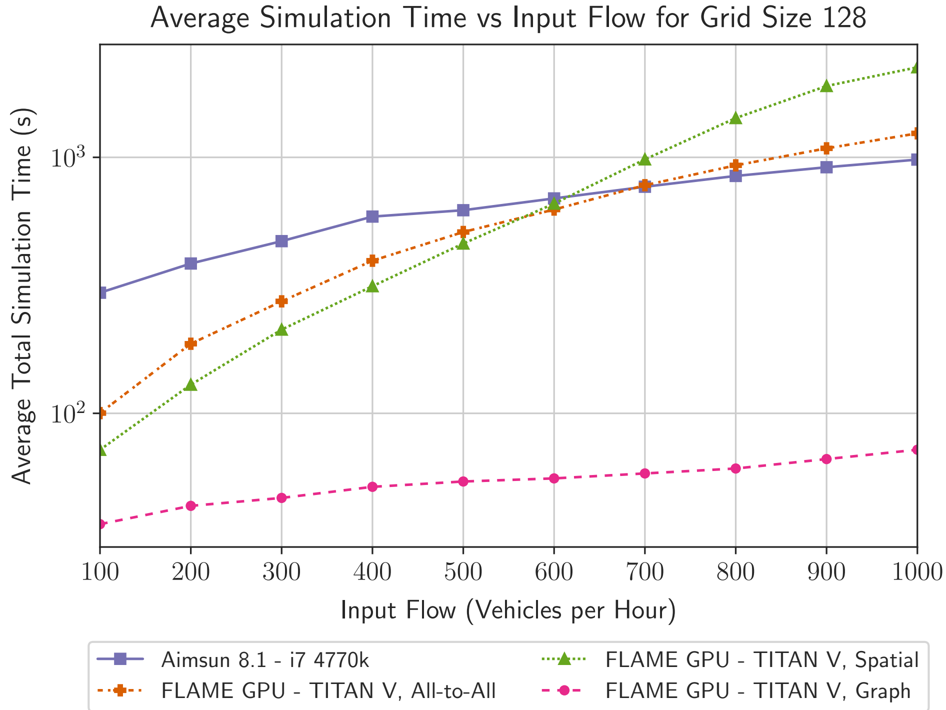


Figure 4.9: Total simulation time against input flow for a procedurally generated road network of grid size 128. Run times shown are the average from 3 repetitions. A logarithmic scale is used for the y axis. The results shown include the reference CPU simulator, Aimsun 8.1, executed on an i7-4770k (purple); and the results from FLAME GPU executed on an NVIDIA Titan V GPU using all-to-all (orange), spatial (green) and graph-based (pink) communication strategies.

following model. This shows the significant reduction in runtime from the graph-based communication strategy compared to the alternate approaches. The magnitude of the runtime is important, as the time taken by this kernel, iterating the message lists, is significantly greater than required to output messages. The improvement for graph-based communication vastly overcomes the additional overhead cost of message output, resulting in a significant overall improvement of performance. The larger reduction in message iteration time is expected, due to the increased granularity of the message lists, and therefore improved work-efficiency when finding the required information.

4.6 Summary

This chapter has expanded on the work Chapter 3 to improve the performance of GPU accelerated road network microsimulations through improvements to the work-efficiency of communication between parallel agents when using message-list based communication, as in FLAME GPU. A method of specialising the communication pattern such that the messages of a message list are associated with an element of the graph is proposed, forming Contribution C2. By associating messages with elements of the graph, less messages must be iterated to find the required information for road network behavioural models such as car following, lane changing

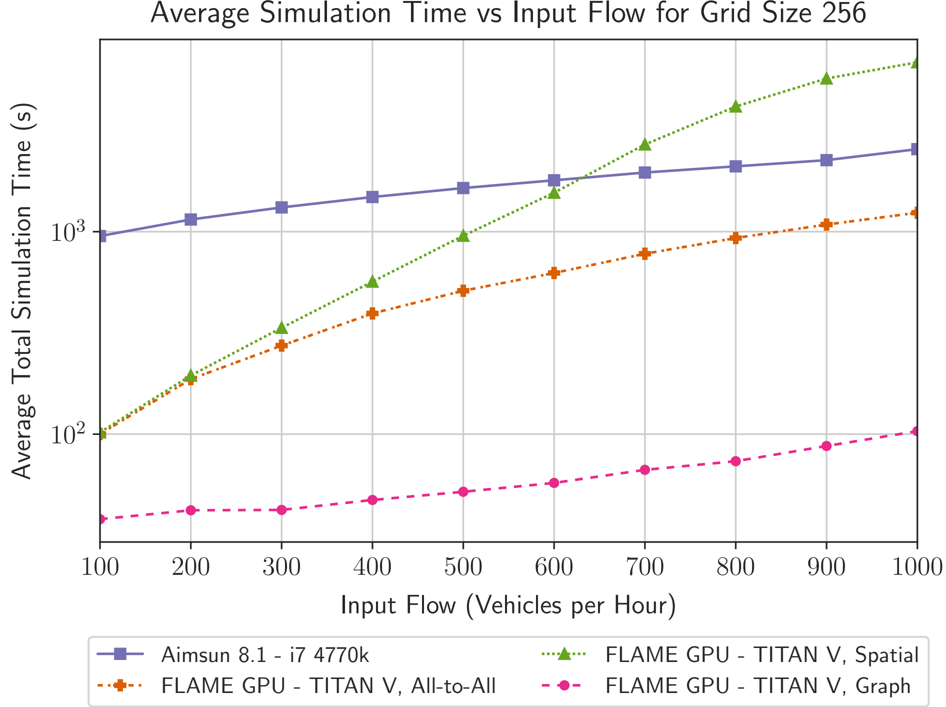


Figure 4.10: Total simulation time against input flow for a procedurally generated road network of grid size 256. Run times shown are the average from 3 repetitions. A logarithmic scale is used for the y axis. The results shown include the reference CPU simulator, Aimsun 8.1, executed on an i7-4770k (purple); and the results from FLAME GPU executed on an NVIDIA Titan V GPU using all-to-all (orange), spatial (green) and graph-based (pink) communication strategies.

or gap avoidance. This method has higher overhead costs in constructing the fine-grained data structure required to access the appropriate messages, but this cost is negligible compared to the potential improvements during message iteration. This proposed strategy is embedded within the FLAME GPU simulation environment, as of version 1.5.0.

An benchmark ABM is proposed within Section 4.4, to support the evaluation of this communication strategy isolated from the complex behaviours of road network simulations. This model is then used to benchmark and evaluate the performance of the communication pattern, demonstrating significant improvements in work-efficiency and performance, and providing Contribution C3 of this thesis.

Subsequently, Contribution C4 is achieved by applying this new communication strategy to the road network microsimulation model from Chapter 3, and demonstrated by the reapplication of the benchmark experiments shown previously. The largest simulation executed using both the CPU simulator (Aimsun 8.1) and the FLAME GPU based simulations, a one-hour simulation containing up to $512,000$ vehicles and $1,575,936$ detectors, shows a maximum speed-up of 67.7x for the GPU accelerated simulation using graph-based communication, with a RTR of 44.7 compared to 0.67 on the CPU. The communication strategy was significant in achieving this level of performance, with the graph-based communication strategy out-performing the

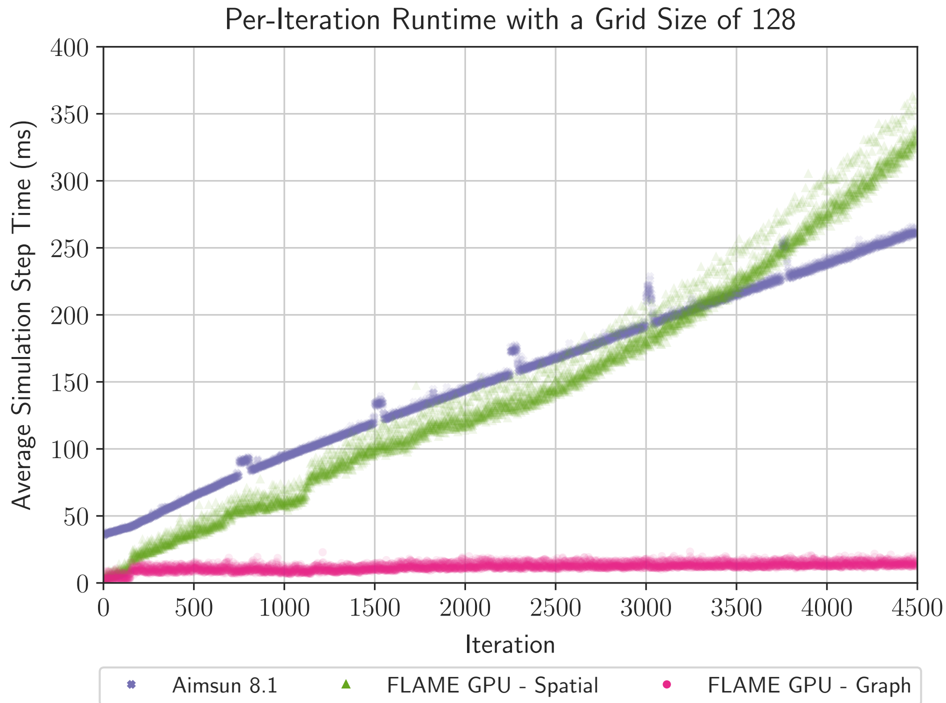
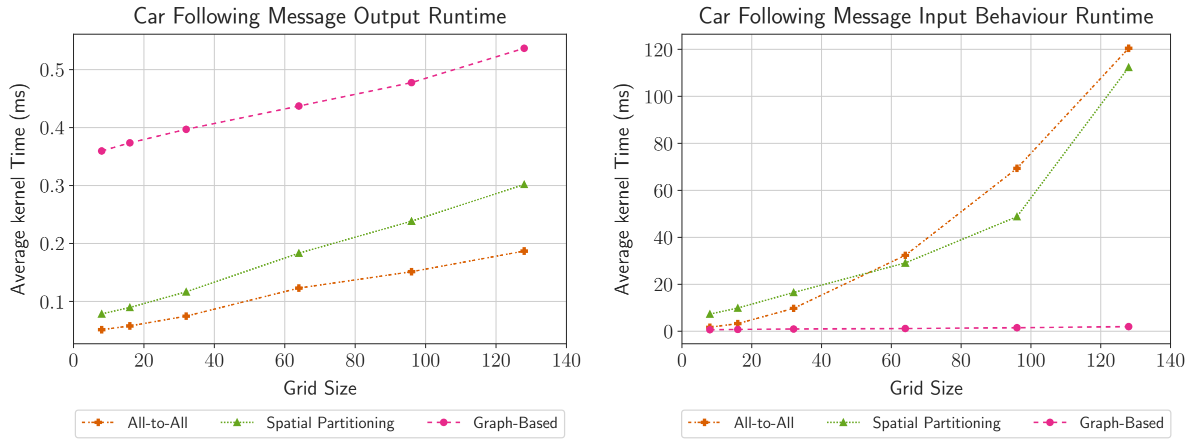


Figure 4.11: Per-Iteration simulation time for a model with grid size 128 and input flow 500 vehicles per hour. Executed using Aimsun 8.1 (purple) on an i7 4770k and using the FLAME GPU implementations with Spatially Partitioned communication (green) and Graph-based communication (pink) on an NVIDIA Titan V GPU.

spatially-partitioned messaging implementation by up to 62x. The scaling behaviour of the graph-based communication is also greatly improved, compared to both CPU and alternate GPU implementations. The largest simulation of $576,000$ vehicles and $1,994,112$ detectors completing in 68% of the time of a much smaller (up to $32,000$ vehicles and $6,336$ detectors) simulation in Aimsun 8.1.

The impact of GPU accelerated simulation has been shown to provide significant performance improvements to microsimulation of transportation systems. In order to understand the broader implications of how this may effect a modellers choice of simulation approach, the impact of GPUs on other modelling and simulation approaches, such as macroscopic simulation, must also be considered. This is the topic of the next chapter.



(a) Message output for each communication strategy, including data structure construction costs.

(b) Performance of message input and execution of the car following model behaviour.

Figure 4.12: The impact of FLAME GPU communication specialisation on the performance of the Car Following Model implementation can be separated into the cost of the message output and the message iteration within the car following agent function.
FLAME GPU was modified to accurately record the cost of each process, and the average of three repetitions at each scale is shown. The simulations were executed on an NVIDIA TITAN X (Pascal) GPU.

Chapter 5

GPU Accelerated Macroscopic Assignment and Simulation

5.1 Introduction

Macroscopic transport network simulations are typically used for the design of transport systems or changes to infrastructure, rather than active management. The insight gained from the simulations inform the decision making process regarding improvements to facilitate the ever-increasing demand on our transport networks, including the optimisation of dynamic control systems [247], [248].

From a simulation perspective, macroscopic simulations are top-down, used to accurately model the flow of transport through a network with a high level of abstraction. This high level of abstraction results in lower computational and data demands compared to alternate, higher resolution techniques. Although computational demands may be lower than alternate approaches, large-scale simulations still take considerable lengths of time to run per simulation. Multiple simulations are often required, for example when evaluating different transport network configurations, evaluating schemes at multiple levels of demand or for scenarios with different weather conditions. The effectiveness of these simulations can be reduced by these computational constraints, as practitioners are restricted in the quantity, variety and scale of simulations which can be completed within the duration of a design cycle [14].

To reduce the run-time of these macroscopic simulations, and enable a greater number of simulations to be completed within a design cycle, greater levels of parallelism must be achieved both within individual simulations and across multiple simulations. Current state of the art for macroscopic modelling and simulation leverage multi-core Central Processing Units (CPUs) to improve simulator performance [234], [238], but there are limits on the degree of parallelism available and the performance scaling as additional processing cores are used [237], [249]. An alternate approach is to use many-core processors such as Graphics Processing Units (GPUs),

which offer significantly higher degrees of parallelism and improved performance characteristics, however, it can be challenging to exploit the massively-parallel nature of GPUs. Algorithms and data structures used within applications used within multi-core CPU transport simulation must be replaced with alternatives which can leverage the highly-parallel hardware. Chapters 3 and 4 have shown that GPUs can be applied to road network modelling and simulation when using a microscopic approach via fine-grained data-parallelism, however, these chapters also showed that there are architectural and algorithmic difficulties in the application of data parallelism to simulation which must be carefully considered. For instance, as presented previously in section 2.1.3, road networks are typically represented by directed weighted graphs within macroscopic road network modelling and simulation. Directed graphs are made up of a set of vertices, with edges forming one-way connections between two vertices. In weighted graphs each edge has an associated cost. Road network graphs often contain zones (also known as centroids), which are a special class of vertex within the graph, used as a mechanism to associate demand data with the road network. Graphs representing transport networks typically have a low average degree and therefore low density with large diameters. These characteristics play an important part in algorithm selection and performance.

Macroscopic road network simulations are often multi-stage processes, which use an iterative process involving the assignment of transport demand onto a road network, followed by a coarse-grained simulation of the road network, which is repeated until some convergence criteria have been met. The assignment phase involves the selection of routes for journeys from an Origin-Destination (OD) matrix onto the road network. The simulation phase then applies the assigned journeys to predict the status of the transport network given the assigned demand. The assignment portion of this iterative process can dominate the simulation run-times.

This chapter presents two main contributions. Contribution C5 is the proposal of a highly-parallel algorithm for concurrently finding the shortest paths through a network from multiple origin vertices - an Many Source Shortest Path (MSSP) algorithm. Unlike existing state of the art GPU accelerated Single Source Shortest Path (SSSP) algorithms [225], [229], [250] which optimise for dense graphs such as social networks, the MSSP algorithm optimises for sparse, high-diameter graphs characteristic of transport networks. To demonstrate the performance impact of the algorithm on macroscopic road network modelling, it is embedded within the Simulation and Assignment of Traffic to Urban Road Networks (SATURN) transport simulation software suite, with the objective of reducing the run-time for large scale macroscopic simulations compared to the existing, highly-optimised multi-core CPU solution. SATURN is a macroscopic assignment and simulation package used for the analysis and evaluation of traffic management schemes [235], used for a broad range of tasks within the transport modelling sector including large-scale regional models of the UK [239]. Embedding the proposed data parallel approach within the SATURN software suite ensures that the performance results reported can

be conceptualised with real world examples rather than abstract, theoretical benchmarks. This evaluation within a macroscopic simulation suite widely used within the UK, using real-world networks provides Contribution C6 of this thesis.

This chapter is organised as follows. Section 5.2 describes the macroscopic assignment and simulation process within SATURN, focussing on the performance limiting algorithms. Section 5.3 presents a set of real-world benchmark road network models of various scales which are used to evaluate the performance of alternate macroscopic road network assignment and simulation implementations throughout this chapter. Section 5.4 describes the novel MSSP algorithm offers improved GPU performance over highly efficient serial SSSP implementations on the CPU, and advances the state of shortest path calculations on the GPU for sparse graphs. Section 5.5 provides details of a novel GPU algorithms which make use of the many shortest routes through a transport network during the assignment process, when embedded within a major commercial macroscopic road network simulation tool. Details of how this may be achieved using multiple GPUs is provided by section 5.6. Section 5.7 discusses the cross-validation of the GPU implementation against the existing CPU-based macroscopic road network assignment and simulation model. Section 5.8 discusses the benchmark methodology and provides the results and a discussion of the benchmarking of the GPU accelerated version of SATURN using real-world road networks, leveraging the advancements described in previous sections to offer improved performance and reduced runtime compared to the existing multi-core CPU-based implementation. Section 5.9 concludes the chapter.

5.2 SATURN

SATURN is a software suite for the analysis and evaluation of traffic management schemes [235]. It was originally developed in 1982 by the Institute for Transport Studies at the University of Leeds as a combined simulation-assignment model. More recently, it has been extended for "pure junction simulation" and "conventional traffic assignment model, with or without simulation" [202]. The SATURN suite contains many applications for different steps in the traffic management scheme evaluation process. *SATALL* is the main application when used as a combined assignment-simulator model. The SATURN user manual [202] describes the broad range of metrics which can be used during the evaluation of a transport scheme.

SATURN currently leverages multi-core CPU parallelism to increase simulator performance, and decrease the time taken for simulations to complete. This is characteristic of other macroscopic assignment and simulation tools, as discussed in section 2.4.1. The commercially-available multi-core implementation of SATALL uses OpenMP [95] to provide task-level parallelism and coarsely-grained data parallelism, using the multiple processor cores to solve a small number of independent tasks concurrently within key portions of the application. The serial version of

SATURN is compiled using the Silverfrost Fortran Compiler [251], whilst the multi-core portion of the parallel implementation is compiled using the Intel Fortran compiler [252].

In order to produce a GPU accelerated implementation of the algorithm implemented within SATALL, the overall process and performance characteristics must first be understood. Figure 5.1 shows the assignment-simulation loop used within SATURN. It uses the Frank-Wolfe algorithm [191] to iteratively converge on a stable state of road network assignment, following Wardrop's principal of traffic equilibrium [190]. Wardrop's principal states that traffic will settle down into an equilibrium where no driver can reduce their journey time by choosing an alternate route. Within this chapter, SATURN version 11.3 is used.

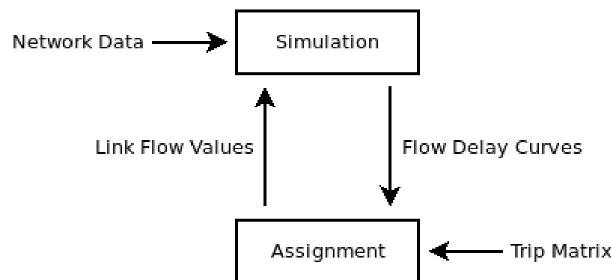


Figure 5.1: The assignment-simulation loop within SATURN [235]

Before applying the convergent assignment-simulation loop, the data used can be prepared to improve the performance of algorithms within the iterative process without impacting the results of the simulation. For instance, the density, diameter and average degree of a graph may influence the performance of algorithms which operate over the graph. As transport networks are typically very sparse graphs with large diameters, it is often beneficial to attempt to find an alternate representation of the network which modifies these properties. *Contraction hierarchies* [199] (previously discussed in Chapter 2) can be used to generate a denser representation of a graph, which can be used within key algorithms, and the results mapped back onto the original, sparse representation. Within SATURN the denser representation of the transport network is known as a *SPIDER* network [198]. The application of the contraction hierarchy involves the modification of the network graph, replacing chains of directly connected edges into a single edge between the origin vertex of the original edge, and the destination vertex of the final edge. The weight of the new edge is aggregated from the weights of the original edges. Essentially shortcuts are created, with a single edge representing a chain of several real edges. If all of the routes which involve a node have been replaced by shortcuts, then for the purposes of assignment the vertex can be removed along with the associated edges. This has the effect of reducing the number of vertices in the graph (and likely edges) resulting in a denser representation, with a smaller diameter. This is a 1:1 mapping, so the results of an assignment or simulation of a denser representation can be directly mapped back on to the original network. Contraction hierarchies have previously been shown to offer significant performance advancements when applied graph

Subroutine	CPU Time (seconds)	Proportion of Total (%)
A	39337.91	97.39
B	135.98	0.34
C	107.59	0.27
D	58.09	0.14
E	57.77	0.14
F	55.59	0.14

Table 5.1: Serial Fortran per-routine run-time performance for large real-world benchmark model (5194 zones) sorted by time. The six longest-running routines are shown but with the mangled subroutine names removed. The most time consuming subroutine A is a key subroutine called within the assignment phase of the assignment simulation loop. This subroutine finds the flow of vehicles per edge in the network, based on the origin-destination information and the state of the network from the previous assignment-simulation loop.

algorithms such as shortest path calculations [199]–[201].

Upon profiling of the serial implementation of SATALL using a large scale real-world network, with typical parameters, a single subroutine is highlighted as accounting for over 97% of a 12 hour runtime (see Table 5.1 for the runtime of the top 6 subroutines). This clearly highlights the area of the application where the majority of the application time is spent, and identifies this area as a candidate for potential GPU optimisation. The subroutine in question, A, corresponds to part of the assignment phase of the assignment-simulation loop. It involves the calculation of the flow of vehicles per edge within the network, given some demand information and the current time taken to traverse each edge within the network, from the previous iteration of the assignment-simulation loop. This process is performed for each user-class within the model (where a user-class is a category of vehicle such as car, motorcycle, Light Goods Vehicle (LGV) or Heavy Goods Vehicle (HGV)) with different demand information and network costs for each user-class.

The per-user-class process can be further divided into two algorithmic processes: (i) *Shortest Path Calculation* and (ii) *Flow Accumulation*. First the shortest path between a pair of zones (origin or destination within the demand matrix, represented by virtual vertices) is found using an SSSP algorithm. This is followed by the accumulation of vehicle flow for each edge in the selected route. In the serial implementation, the shortest path calculations account for 95% of the subroutine runtime (92% of the total runtime). The overall process for assigning demand to a network as flow is shown in algorithm 4.

Section 5.2.1 and section 5.2.2 describe the respective algorithms in more detail.

5.2.1 Shortest Path Calculations

The sequential and multi-core implementations of SATALL make use of the *D'Esopo-Pape* SSSP algorithm [212] to find the routes between a single origin zone and all other vertices (including zones). D'Esopo-Pape is a work-efficient but highly sequential algorithm. This makes it well-

Algorithm 4 Existing Serial SATALL algorithm

Input: Graph G with Z zones,
UC User classes,
Origin-Destination Matrix OD of dimensions Z by Z

Output: Flow per edge in G, F

1: **for each** User-Class **do**
2: **for** origin zone $o \in \{1, ...Z\}$ **do**
3: $paths \leftarrow$ call $CalculatePaths(o, G, OD)$
4: **for** destination zone $d \in \{1, ...Z\}$ **do**
5: $F(o, d) \leftarrow$ call $AccumulateFlow(d, o, paths, G, OD)$
6: **end for**
7: **end for**
8: call $AggregateAndPostProcessFlowData(F)$
9: **end for**

suited to sequential or task-parallel approaches, but not appropriate for implementation using data-parallelism.

The D'Esopo-Pape algorithm was selected for implementation within SATURN as it outperformed other SSSP implementations at the time of development for the scale of network being simulated at the time [198]. The algorithm is shown and described within the literature review in section 2.4.2.1. An alternative work-efficient SSSP algorithm which may offer increased CPU performance is *Dijkstra's* algorithm [215], described in section 2.4.2.1. The most performant variant of Dijkstra's algorithm uses a Fibbonaci Heap [216]. This is asymptotically the most efficient serial SSSP algorithm. However, the scale of improvement offered would likely be relatively small, as shown in literature comparing SSSP implementations [223], and any performance improvement would also be highly dependent on the network in question [221].

An alternative to using an SSSP algorithm would be to use an All Pairs Shortest Path (APSP) algorithm, as presented in section 2.4.2.2. However, this would be highly inefficient as the assignment process only requires the shortest paths between the zones within the network, rather than paths between all vertices. As the number of zones is typically only a small proportion of the number of vertices within the network, even after contraction hierarchies have been used to create a denser representation, many unnecessary paths would be found by APSP algorithms. This would require additional compute resources, but more significantly increased memory resources to calculate and store the additional routes and costs. As such this class of algorithm were dismissed, for the case where paths from "many" but not all origins are required. Instead a novel data-parallel shortest path algorithm to find the paths from many origins, based on the Bellman-Ford SSSP algorithm is proposed in section 5.4.

5.2.2 Flow Accumulation

The calculated shortest paths through the road network are used to map vehicle flow onto the transport network from the origin-destination matrix. The sequential algorithm, used in the serial and task-parallel multi-core implementations of SATURN, is straight-forward and performs relatively well, as described in Algorithm 5. For a given origin zone, this algorithm iterates the destination zones within the matrix, determines if the origin and zone are connected, and if so traces the route along the shortest path from the destination zone back to the origin. As each edge is encountered, the trip cost for the origin-destination pair is accumulated onto the total flow assigned to that edge.

Algorithm 5 Sequential Flow Accumulation

Input:

Z is the number of zones,
G is Graph,
O is origin zone,
P are Shortest paths from all zones,
OD is Origin-Destination Matrix of dimensions Z by Z F is Flow per edge in G, from the previous Origin

Output:

F is Flow per edge in G

```
1: for destination zone d ∈ {1, ...Z} do
2:   f ← OD(O, d)
3:   e ← call PredecessorEdgeFromVertex(O, d, P, G)
4:   while e < |G_E| do
5:     F(e) ← F(e) + f
6:     v, v′ ← e
7:     e ← call PredecessorEdgeFromVertex(o, v, P, G)
8:   end while
9: end for
```

5.3 Real-World Benchmark Models

To evaluate the performance impact of using alternate or novel algorithms within SATURN, executed on different hardware architectures, a set of benchmark road networks are required. Rather than use a set of procedurally generated models to benchmark the algorithmic performance, a set of real-world networks in use within the UK were used as benchmark models, ranging from very small town-scale models to large region-scale models including metropolitan areas. Table 5.2 provides relevant properties of the networks, including the number of user classes, number of zones, as well as the vertex and edge counts for both original and spider representations of the network. The *Epsom* model is a very small model provided as an example within the SATURN suite, modelling a town within Surrey in England. The second model, *Derby* is a model of the city of Derby, England. The CLoHAM (Central London Highway

Network	User Classes	Zones	Original Vertices	Original Edges	Spider Vertices	Spider Edges
Epsom	2	12	89	130	17	74
Derby	13	547	19044	29081	2700	25385
CLoHAM	5	2548	80609	147481	15179	132600
LoHAM	5	5194	129330	196546	18427	192711

Table 5.2: Properties of the real-world SATURN networks used during development and benchmarking. The number of vertices and edges are shown for both the original and Spider (contraction hierarchy) variants.

Assignment Model) and LoHAM (London Highway Assignment Model) models are strategic models for motorised trips using London's highway network, developed by Transport for London (TfL) [253], [254]. The properties of these real-world networks in table 5.2 demonstrate the low density which is characteristic of road network graphs, for instance the largest network, *LoHAM*, has a density of 1.18×10^{-5} in it's original form and a density of 5.68×10^{-4} using the denser SPIDER representation.

The smallest model, *Epsom*, contains only 12 zones, 89 vertices and 170 edges in the original representation, making it a very small network. As such it is too small to justify GPU execution and is excluded from performance related figures and tables.

5.4 GPU Many Source Shortest Path for Sparse Graphs

Within an iteration of the assignment-simulation loop of SATURN, the shortest paths must be calculated between each origin and destination zone, for each user class within the simulation. Each user-class can be treated as a separate independent process for the shortest path computation, although multiple user-classes can be processed concurrently on a single GPU.

The proposed shortest path algorithm is based on the Bellman-Ford SSSP algorithm [213], but heavily modified to improve performance for low-density, high-diameter graphs characteristic of road networks. The most significant conceptual change compared to the traditional Bellman-Ford algorithm is that the algorithm solves the shortest path problem for multiple origin vertices concurrently, rather than only a single source vertex, i.e. it is a MSSP algorithm. Furthermore, unlike APSP algorithms it does not necessarily solve for *all origins* either (although it could be used to do so). For the LoHAM network, the MSSP algorithm operating over all origin zones will find 28% of possible shortest paths within the graph when using the denser SPIDER representation, or only 4% of the shortest paths using the un-modified graph. This is a substantial reduction in work-load and storage compared to an APSP algorithm.

As detailed in section 2.4.2.1, the core Bellman-Ford algorithm is a relaxation-based algorithm, which iteratively refines approximations of the shortest paths between the origin vertex and all other vertices until the minimal-cost paths are found. At each iteration, each edge within

the graph is considered for relaxation. If the edge can result in a lower cumulative cost from the origin to destination vertex of the edge, the estimate of cost and edge used are updated for the destination vertex. This is repeated for $|V| - 1$ iterations, ensuring that all shortest paths have been found.

Several of the common modifications to the Bellman-Ford algorithm are included to improve the work-efficiency of the algorithm when considering a single source. Firstly, early-termination of the algorithm can be achieved in most cases, by detecting if any changes have been made during the iteration. The algorithm can be terminated when no changes occur within an iteration [224].

The work-load can further be reduced using the *vertex-frontier (VF)* approach [225], [226]. The vertex frontier approach is based on the fact that during the first iteration of the Bellman-Ford algorithm, only edges which leave the origin vertex will be relaxed successfully. In subsequent iterations, only edges leaving vertices which were updated during the previous iteration will be relaxed. By tracking the vertices which were updated by relaxations in a given iteration, within the vertex frontier, the work-efficiency can be improved as fewer edges are be considered at each iteration. The vertex frontier must be initialised to contain the origin vertex, and the iterative process can stop once the vertex frontier is empty. For instance, Figure 5.2 shows a simple graph containing 9 vertices and 10 edges which can be used to demonstrate the use of the vertex-frontier based approach. Figure 5.3 shows the contents of the vertex frontier at each iteration of the algorithm when finding the routes from origin vertex `a`. The vertex frontier is initialised to only contain the origin vertex `a`. During the first iteration, each edge leaving vertex `a` is relaxed, resulting in updated route costs for vertices `b`, `c` and `d`, which are stored in the vertex frontier for the second iteration of the algorithm. In the second iteration, the three edges leaving vertices in the frontier are relaxed, resulting in three updated cumulative costs estimates and a new vertex frontier containing vertices `e`, `f` and `g`. The third iteration results in three new cumulative cost estimates, but the frontier only contains the elements `h` and `i` as there is no need to store the same element (`i`) multiple times. Subsequently, the fourth iteration results in a single update and a new frontier of a single element, followed by an empty frontier at the end of fifth iteration. As the frontier is now empty, there is no more work to be done and all shortest paths from the origin vertex `a` have been found.

Graphs representing transport networks are typically very sparse, with low-density and high-diameter. This results in relatively small vertex frontiers, especially during early iterations as the frontier is growing, and during later iterations, due to the high-diameter of road network graphs. The size of the vertex frontier directly impacts the level of parallelism available within the algorithm, resulting in under utilisation of modern GPUs. For instance, full occupancy of the NVIDIA Titan V GPU requires $163,840$ threads when there are no resource constraints which impact occupancy [79]. This lack of parallelism limits the achieved performance of the

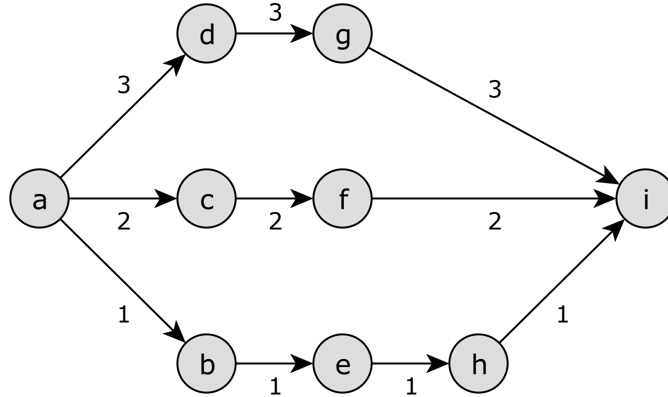


Figure 5.2: A simple example graph to illustrate the use of a vertex-frontier.

Iteration	Frontier Vertices
0	a
1	b c d
2	e f g
3	h i
4	i
5	

Figure 5.3: The contents of the vertex-frontier for the Bellman-Ford algorithm, for origin vertex `a` of the example graph in Figure 5.2

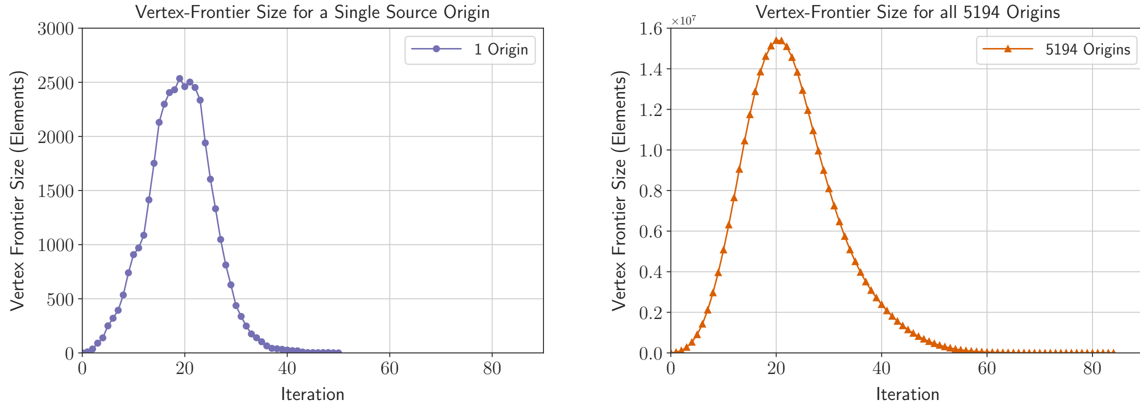
SSSP algorithm implemented for the GPU, achieving reduced performance compared to CPU implementations of highly work-efficient SSSP algorithms such as Dijkstra's algorithm.

To address the lack of parallelism within the Bellman-Ford shortest path algorithm when processing sparse road networks, and given that the paths from many source vertices are required within road network assignment, a novel modification to the vertex-frontier based Bellman-Ford algorithm is proposed. The *Origin-Vertex Frontier (OVF)* is an extension to the Vertex-Frontier concept. Each element within the OVF contains both the recently updated vertex, and the origin which it corresponds to as a pair. The algorithm using the origin-vertex frontier is similar to the simpler vertex-frontier based algorithm. The OVF is initialised to contain the origin vertex for each origin being considered. While the OVF is not empty, the edges which leave each element of the OVF are relaxed, with updated origin-vertex pairs being stored within the next OVF. Once the OVF is empty, all shortest paths for the set of origin vertices have been found. This switch from an SSSP algorithm to MSSP algorithm via the OVF forms Contribution C5 of this thesis.

For instance, using the graph from the previous example (figure 5.2) to find the paths from origin vertices `a` and `b`, the OVF would be initialised as `(a, a), (b, b)`. Figure 5.4 illustrates the contents of the OVF as the iterative algorithm progresses. After the first iteration, the OVF would include origin-vertex pairs corresponding to updates caused by edges leaving `(a,a)`

Iteration	Frontier Origin-Vertex pairs
0	`(a, a), (b, b)`
1	`(a, b), (a, c), (a, d), (b, e)`
2	`(a, e), (a, f), (a, g), (b, h)`
3	`(a, h), (a, i), (b, i)`
4	`(a, i)`
5	

Figure 5.4: The contents of the origin-vertex-frontier within the the MSSP algorithm, for origin vertices `a` and `b` of the example graph in Figure 5.2



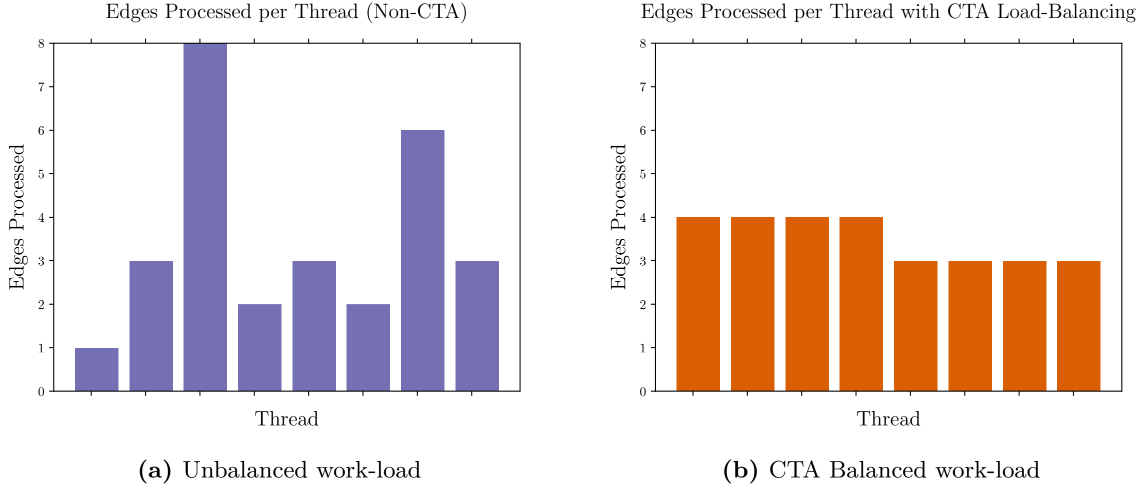
(a) Vertex Frontier Size for a Single Origin Vertex **(b)** Origin-Vertex Frontier Size for All Origin Vertices

Figure 5.5: The size of the vertex-frontier and origin-vertex-frontier at each iteration of the modified Bellman-Ford algorithm for the LoHAM large road network, containing 5194 zones. The frontier size is shown for a single origin vertex and for all 5194 origin vertices.

and `(b,b)`, resulting in a new origin-vertex frontier of `(a, b), (a, c), (a, d), (b, e)`. Following a further iteration the OVF would contain `(a, e), (a, f), (a, g), (b, h)`, and so on until the OVF is empty.

This can cause a significant increase in the size of the frontier (if sufficient numbers of origins are being considered), and therefore parallelism exposed to the GPU's many processing cores. Figure 5.5 shows both the vertex-frontier for a single origin, and the origin-vertex-frontier for the largest real-world road network being considered, LoHAM, for all 5194 origin zones. The OVF for 5194 origins is significantly larger than the single origin VF, indicated by the significantly different scale of the y-axis. The OVF is sorted by both origin index and vertex index at each iteration. This is a key optimisation to ensure good utilisation of memory bandwidth and cache efficiency by maximising the likelihood of memory coalescence by improving data locality.

For the GPU implementation of this algorithm, other implementation-specific factors are also considered. Within the road network graph the degree of vertices varies over a small range. This can result in an imbalanced work-load if each thread considers a single vertex, as threads will process different numbers of edges. This can have a negative impact on performance on Single-Instruction-Multiple-Thread (SIMT) architectures, as some threads (and warps) will be idle whilst higher-degree vertices are still being processed.



(a) Unbalanced work-load

(b) CTA Balanced work-load

Figure 5.6: The load-balancing effect of the cooperative thread array for 28 edges distributed across 8 threads. The imbalance of work has been minimised from a maximum divergence of 7 edges to a single edge.

Further improvements to performance can be found by using a Cooperative Thread Array (CTA)[255] to balance the workload between threads. Vertices in the OVF may have differing degrees to one another. If individual threads process all edges which leave the vertex, then divergence will occur based on the degree of the vertex being processed. This can be balanced using a CTA, so that each thread processes a fair share of the edges within the block of threads, reducing divergence within the block. The cooperative group of threads must first find the total number of edges they are collaborating on, based on the number of edges leaving each vertex assigned to the block of threads, storing this in low-latency shared-memory. Once this is known, each thread can then access a single edge from the pool of edges to relax. This can be efficiently implemented using a binary search. Threads will iteratively process a single edge from the pool, until all edges for the block have been processed. Figure 5.6 illustrates this load balancing approach. Figure 5.6a shows the work-balance (and approximate run-time) per thread for an example case with imbalance. When using a CTA, the variance in edges processed per thread is reduced, as illustrated by figure 5.6b. In this case the imbalanced is reduced, with neighbouring edges performing the same amount of work. When scaled up to a full block made up of multiple warps, this resolves imbalance within warps, whilst also minimising divergence between warps. Using the CTA does impose some additional cost in both compute and memory bandwidth, however, the overhead costs are not significant compared to the improvements gained through the reduction in divergence.

Algorithm 6 details the multiple source shortest path algorithm, based on the Bellman Ford algorithm. In the data-parallel implementation of this algorithm, a single GPU kernel is executed to initialise the result arrays (lines 5-14), reducing kernel launch overhead. The for loop used to initialise the origin vertex frontier OVF, lines 15-17, and the per-OVF element for loop,

lines 20-32, are also parallelised using CUDA kernels. The inner while-loop used to process edges leaving vertices in the OVF (lines 21-31) is implemented using a CTA. This is implemented using shared memory within a block of threads. A binary search is used to select the next edge from shared memory to load-balance the non-uniform degree of vertices within the graph. A single atomic transaction is used to update the results arrays on lines 25-26 to avoid race conditions. Atomic operations have higher latency than non-atomic global memory operations, as the read-modify-write operation includes the latency of two global memory operations. The latency can also be compounded by the serialisation of atomic memory operations to the same address by competing threads (atomic contention). For recent NVIDIA architectures, the additional latency compared to non-atomic operations has been measured in the order of 10s of cycles for up to 32 contentious operations [256]. Although the additional latency from the use of atomic operations is not ideal, it is less-impactful than alternative methods of avoiding race conditions.

5.5 GPU Flow Accumulation

Within the CPU implementation, the flow accumulation algorithm is relatively simple with an insignificant impact on runtime compared to the shortest path calculations. However, with the GPU implementation of the shortest path calculation, the cost of transferring the shortest path results to the host for serial computation (or parallel CPU-based computation) takes a relatively large amount of time, potentially offsetting any benefit of computing the shortest path results on the GPU. As such it is preferable to minimise the costly device to host memory transfers, and consider a GPU implementation of the flow accumulation process, as the volume of data which must be transferred after this process is much smaller than that produced during the shortest path calculation phase. In some use-cases however, such as select link analysis, the data must be transferred back to the host. This will typically only occur once the simulation has converged, in which case the impact on performance is less significant.

As described in section 5.2.2, the flow accumulation process traces routes from each destination zone back to each origin zone in the OD matrix, adding the demand on to each edge of the route. For a parallel implementation, care is required when writing to the shared results array, which for a many-core GPU can be achieved using atomic addition.

For the purposes of SATURN, double-precision floating point numbers are used for trip costs and edge-flow values, as a large range of demand values can be used and the accumulation of these values can result in a loss of value if lower precision numbers are used.

Algorithm 7 describes a relatively naive approach to accumulate these flow values in parallel. Each trip from the origin-destination matrix (for each user-class) is handled by an individual CUDA thread. Each CUDA thread follows the route from the destination, back to the origin one edge at a time. As each edge is visited, the flow value for the trip is added to the flow value for

Algorithm 6 Many Source Shortest Path (MSSP) algorithm based on the Bellman-Ford algorithm, using an Origin-Vertex Frontier for execution on SIMT architectures.

Input:

V is the number of vertices,
E is the number of edges,
Z is the number of zones,
G is Graph with edge costs C,
OD is Origin-Destination Matrix of dimensions Z by Z

Output:

BE is Back-Edge array containing one element per vertex per origin,
BC is Back-Cost array containing one element per vertex per origin

1: Let OVF be the Origin-Vertex Frontier, a set of origin-vertex pairs
2: Let NF be the Next Origin-Vertex Frontier, a set of origin-vertex pairs
{Initialise the frontiers to empty}
3: $OVF \leftarrow \varnothing$
4: $NF \leftarrow \varnothing$
{Initialise the Back-Edge and Back-Cost arrays, using one thread per array element}
5: **for each** $idx \leftarrow 1$ to $V \times Z$ **do**
6: Let $v \leftarrow idx \div Z$
7: Let $o \leftarrow idx \bmod Z$
8: **if** $o = v$ **then**
9: $BC(o, v) \leftarrow 0.0$
10: **else**
11: $BC(o, v) \leftarrow \infty$
12: **end if**
13: $BE(o, v) \leftarrow |G_E| + 1$
14: **end for**
{Initialise the origin-vertex frontier with one element per zone}
15: **for each** $idx \leftarrow 1$ to Z **do**
16: $OVF.append(\{idx, idx\})$
17: **end for**
{While the origin-vertex frontier is not empty, iterate the bellman-ford algorithm}
18: **while** $OVF \neq \varnothing$ **do**
19: $NF \leftarrow \varnothing$
{For each element of the origin vertex frontier, using 1 thread per element}
20: **for each** $idx \leftarrow |OVF|$ **do**
{Co-operatively select an edge from the pool of vertex-frontier edges}
21: **while** $e \leftarrow$ edgeFromCooperativePool(idx) **do**
22: $v, v' \leftarrow e$ {Source and destination vertices from edge}
23: $c \leftarrow C(e)$ {Cost of edge}
{If the edge results in a lower cost to the vertex, atomically update}
24: **if** $BC(v) + c < BC(v')$ **then**
25: $BC(v') \leftarrow BC(v) + c$
26: $BE(v') \leftarrow e$
{If the next vertex is not a zone, insert into the next frontier}
27: **if** $v' > Z$ **then**
28: $NF.append(\{o, v'\})$
29: **end if**
30: **end if**
31: **end while**
32: **end for**
33: **end while**

the edge (stored in global memory) using an atomic operation to avoid any race conditions. Due to the characteristics of road networks which typically contain trunk roads such as motorways or A roads, which many routes will traverse, this can lead to very high *atomic contention* for certain elements of the edge-flow array. This can result in a loss of performance, especially on older GPUs (Maxwell generation and older) which do not have a hardware-level instruction for atomic addition of double-precision floating point numbers.

Algorithm 7 GPU Flow Accumulation using double-precision atomic addition. This is suitable for Pascal-generation Nvidia GPUs and newer.

Input:
  Z is the number of zones,
  G is Graph,
  P are Shortest paths from all zones,
  OD is Origin-Destination Matrix of dimensions Z by Z
Output:
  F is Flow per edge in G
  {For each trip in the matrix, in parallel using one thread per trip}
1: **for each** trip t from origin o to destination d **in** OD **do**
2:   $f \leftarrow OD(t)$
3:   $e \leftarrow$ call PredecessorEdgeFromVertex(o, d, P, G)
  {While the next edge in the trip is valid, accumulate flow}
4:   **while** $e \leq |G_E|$ **do**
5:     $F(e) \leftarrow F(e) + f$ {**atomic**}
6:     $v, v' \leftarrow e$
7:     $e \leftarrow$ call PredecessorEdgeFromVertex(o, v, P, G)
8:   **end while**
9: **end for**

To improve performance on GPUs which must use a software-based atomic addition operation, a more complex algorithm described in Algorithm 8 can be used which reduces the number of atomic operations, at the cost of a significant amount of additional work. Following this approach, an iterative process of sorting and reductions are used to process the routes in small steps using a frontier-based approach. Initially, a trip-frontier is defined with all valid trips from the OD matrix. This excludes trips with zero demand, or trips to and from the same zone. The trip-frontier contains the trip identifier, and the edge which is next to be processed, initially set to the first edge for that trip. The initial trip-frontier is then sorted by edge, to enable neighbouring threads to co-operate with one another. While the trip frontier is not empty, an iterative process is performed tracing one step of the route at a time until all trips have been completed.

When implemented in parallel for the GPU, a kernel is launched using one thread per trip in the trip-frontier. Each thread loads the trip cost and next edge for the route from global memory. Threads within each block then cooperatively perform a block-level reduction of flow values per edge within the block. The reduced flow value per edge is then atomically added to

the edge-flow array in global memory, using only a single atomic operation per edge within the block. In the worst case, this has one atomic operation per thread, but only a single atomic operation per edge in the best case. Finally, the next trip frontier is sorted by edge, and the total number of trips counted for the next stage of this iterative process.

Although this achieves the goal of reducing the number of double-precision atomic operations required, it does so with significant additional costs compared to the previously described more-naive approach.

Algorithm 8 GPU accelerated flow accumulation algorithm using a reduced number of double-precision atomic operations. This is suitable for hardware with poor double-precision atomic addition performance, such as Maxwell and Kepler generation Nvidia GPUs.

Input:
Z is the number of zones,
G is the Graph ,
P are the shortest paths from all zones,
OD is the Origin-Destination Matrix of dimensions Z by Z
Output:
F is the flow per edge in G

```
1: Let TF be the trip-frontier, a set of trip-edge pairs
2: Let NF be the next trip-frontier, a set of trip-edge pairs
3: Let BLF be local flow value within shared memory at the block level {Initialise the trip-
   frontier TF with all valid trips in the OD matrix}
4: for each trip t from origin o to destination d in OD do
5:    e ← call get-edge-from-path(o, d, P, G)
6:    if e ≤ |G_E| then
7:       TF.append({t, e})
8:    end if
9: end for
   {Sort the trip frontier so edges are in the same block of threads}
10: parallel sort TF by e
   {Trace each trip in parallel, one edge at a time}
11: while TF ≠ ∅ do
12:    NF ← ∅ {Reset the next frontier}
       {Update the flow for each trip in the frontier}
13:    for each (t, e) in TF do
14:       f ← OD(t)
15:       e′ ← call PredecessorEdgeFromVertex(o, d, P, G)
          {Update the next trip frontier if required}
16:       if e′ <= |G_E| then
17:          NF.append({t, e′})
18:       end if
          {Accumulate flow for edge at the block level}
19:       BLF(e) ← blockReduce(f)
20:       F(e) ← F(e) + BLF(e) {atomic}
21:    end for
22:    parallel sort NF by e
23:    TF ← NF
24: end while
```

5.6 Multi-GPU Implementation

The GPU accelerated implementation of many-source shortest path has relatively high memory requirements, both to store the back edges and back edge costs, as well as to store the current and next origin-vertex frontiers. Combined with the fixed quantity of memory available on a GPU, this can limit the number of concurrent user-classes which can be processed on a single device at the same time. Multiple GPUs can be used to combat this, where the GPUs collaboratively work on the same problem. This increases the total memory available, whilst also increasing the total compute available, at the cost of complexity in orchestrating the distribution of work across GPUs.

Within SATURN, and many other transport simulation tools, multiple types of vehicle (user classes) are considered. Road networks often contain sections of road which are limited to certain classes of vehicle (i.e. bus lanes), or different rules such as speed limits for different vehicle classes such as HGVs. This can mean that the shortest path in terms of time from point A to point B may be different for different user classes, and therefore the shortest path algorithm must be executed independently for each user class modelled, before results are combined. Effectively, user-classes form independent chunks of work, each using the MSSP algorithm to compute their independent shortest paths and impact on vehicle flow within the network during the iteration of the assignment-simulation loop. These user-classes may be processed concurrently on a single GPU, or split across multiple devices. This can then be load-balanced so each GPU in the system is performing approximately the same amount of work. For instance if there are 6 user classes and 2 GPUs, each GPU can be assigned 3 user classes.

A small additional step is required to accumulate the flows from each device, where the per-edge flow values from each user-class or each device must be collated, involving some memory transfer and a per-edge accumulation.

Further load-balancing for imbalanced configurations (such as 5 user classes and 2 GPUs) was considered. However, when splitting a user-class into smaller units for independent processing, an additional step is required to combine the independently generated edge-flow values after flow accumulation has been completed on each segment of work. This result combination step includes a relatively large memory transfer of data to consolidate the results on a single GPU. In practice the time required for the additional memory movement and result combination outweighed the benefits and time saved from the more balanced work load across multiple GPUs. Subsequently, this approach was dismissed. Alternatively, some degree of multi-device load balancing may be possible by carefully assigning the user-classes onto devices to balance the total work performed, although this was not pursued.

5.7 Validation

To ensure that the performance of the GPU accelerated implementation can be fairly compared against the existing CPU implementation, the implementation was cross-validated following two approaches. First, the results of the shortest path algorithm for all origins, and the accumulated per-edge flow values were captured from the first iteration of the assignment-simulation loop within the serial CPU version of the SATALL application as a reference case. These results were then compared directly against the shortest path results and accumulated per-edge flow values generated by a single iteration of the GPU implementation of SATALL. Secondly, full runs of the SATALL application were performed, until the convergence parameters of the assignment-simulation loop were met. The metrics selected for validation were the same as used in the existing calibration and validation procedure for the LoHAM network [249].

Alternate shortest path algorithms should produce the same cumulative costs as one another for each vertex within the network (excluding variance from non-commutative floating point precision operations). However, a given network may have multiple routes between two vertices with the same cost. In cases where there are multiple equivalent cost routes, different algorithms may select different routes, subject to the order in which edges are relaxed. Priority queue based SSSP algorithms (such as Dijkstra's algorithm or the D'Esopo-pape algorithm used within the CPU version of SATALL) gain work-efficiency by processing vertices and edges of the graph in deterministic orders, and therefore may generate alternate routes to one another. In contrast, the parallel MSSP algorithm for GPUs does not perform relaxations in a specific order within an iteration of the algorithm, conceptually all relevant edges are processed concurrently at each iteration, potentially leading to alternate route selection. Additionally, the order of execution of threads is not guaranteed within the SIMT programming model of NVIDIA GPUs. This can lead to non-determinism between multiple invocations of the same algorithm even on the same graph. The iterative nature of the assignment simulation can lead to divergence between runs, as minor differences in early iterations can compound into larger and larger variations as the iterations progress, however, as the algorithm is a convergent process the final results should be equally valid.

The results captured from the isolated shortest path and flow validation were an exact match for the smaller models of Epsom and Derby. For the larger CLoHAM and LoHAM networks, discrepancies in the selected shortest path routes were observed, but the cumulative costs of all routes were correct. This highlighted non-determinism within the MSSP implementation itself, due to the use of atomic operations when selecting the shortest edge back to a given vertex. Although the selected paths were correctly shortest paths, determinism is often preferred by application users to ensure results are reproducible. The MSSP novel algorithm described in this chapter can be implemented as a deterministic algorithm by using a more complex atomic

Measure	CPU (%)	GPU (%)	Delta (%)
Links - GEH < 5	64	64	0
Links - GEH < 7.5	78	78	0
Links - DMRB Flow Criteria	74	74	0
Screenline - Flow Difference < 5	90	90	0
Enclosure - Flow Difference < 5	94	96	+2
Mini screenline - GEH < 5	91	91	0
JT Routes - Time Difference < 15	92.1	92.6	+0.5

Table 5.3: Validation metrics for the multi-core CPU and GPU implementations of the LoHAM large-scale model.

operation when relaxing an edge, which considers both the cost to the vertex, and the index of the edge used. This may slightly increase the number of iterations required by the algorithm, but results in deterministic results. The accumulated per-edge flow values, calculated by the flow-accumulation process based on the shortest paths through the network, showed equivalent results to the generated reference data when the same shortest paths were provided, with some floating point variance due to numerical instability and out-of-order execution.

The second validation method of the full SATURN implementation involved the collection of key metrics from the files generated. Table 5.3 contains the high-level validation results from the key shows the values of the key metrics for the largest model used, LoHAM, based on the existing calibration and validation procedure defined for the LoHAM and CLoHAM models [257]. All metrics fall within 2% of the CPU for the GPU implementation. The GEH statistic, first proposed by Geoffrey E. Havers [258], allows sets of traffic volumes to be compared. Within SATALL, when comparing assigned traffic volumes against observed traffic volumes, edges with GEH values of less than 5 are considered acceptable [202]. This measure is an empirical measure used within road network simulations, rather than a true statistical test. The DMRB flow criteria is based on formula from the Design Manual for Roads and Bridges [259], which are used to evaluate weaving for motorway segments within SATALL [202]. Screenline analysis metrics fall under the Select Link Analysis feature within SATURN. Screenlines are virtual boundaries which intersect sections of road. The flow of vehicles across the screenline can be compared to observed data. The proportion of routes with journey times below a threshold are also used to evaluate the implementation.

5.8 Benchmark Results

To measure the performance of the parallel MSSP algorithm, a set of benchmark experiments were performed, using the existing serial CPU implementation, existing multi-core CPU implementation and the new GPU implementation. Three networks of different scales were benchmarked, previously described in section 5.3, across a diverse range of hardware. Table 5.4

	CPU			GPU			
ID	**Model**	**Cores**	**Freq(GHz)**	**Model**	**Cores**	**Freq(GHz)**	**FP64 Ratio**
DC	2x Xeon E5-2667	24	2.9				
P	i7-6850K	6	3.6	2x Titan Xp	3840	1.405	1:32
V	i7-6850K	6	3.6	3x Titan V	5120	1.200	1:2
T	i7-6850K	6	3.6	1x Titan RTX	4608	1.350	1:32

Table 5.4: The hardware used to benchmark real-world performance. The FP64:FP32 Ratio column of the GPU properties describes the ratio of double precision floating point (FP64) units within the GPU compared to single precision floating point(FP32) units. Most GPU architectures include a relative low number of FP64 units, as they are often not used for computer graphics applications. For General Purpose Computing on Graphics Processing Units (GPGPU) codes which may make heavy use of FP64 operations, including the flow-accumulation phase of SATALL, choosing a GPU architecture with a higher ratio of FP64 units can have a significant impact on performance.

Hardware	**Implementation**	Total Runtime (s)			Assignment Runtime (s)		
		Derby	**CLoHAM**	**LoHAM**	**Derby**	**CLoHAM**	**LoHAM**
P	Serial	1202.6	11411.5	41591.9	779.6	11154.4	40974.6
P	Multi-core (6c)	288.1	1540.5	4992.4	128.5	1282.3	4395.5
DC	Multi-core (24c)	361.3	1228.0	3426.1	155.6	893.2	2633.2
P-1	1 GPU	137.3	604.5	1327.5	51.8	455.8	1008.6
V-1	1 GPU	123.9	439.2	903.0	41.8	289.8	584.5
T-1	1 GPU	119.6	447.6	953.7	39.8	298.7	635.7
P-2	2 GPUs	120.8	404.9	864.8	35.6	256.5	547.0
V-2	2 GPUs	115.8	327.9	667.2	32.8	178.6	349.7
V-3	3 GPUs	110.1	303.6	624.1	27.7	154.2	306.1

Table 5.5: Average runtimes in seconds for the total and assignment portions of SATALL application to complete across the three real-world road networks used for benchmarking, across a range of hardware. Note: Serial results are from a single run rather than an average due to runtime.

provides identifiers for each hardware combination used. Each simulation was repeated 3 times to produce an average runtime and remove variance due to external factors such as hardware temperature. This also accounts for non-determinism which may be present in simulations which have multiple, equivalent cost shortest path routes. This allows the fair comparison of the performance of the alternate implementations.

The results of the benchmarking process described above are shown by the tables and figures within this section. Table 5.5 provides the average total processing time, and average assignment processing time for each real world road network using the various software implementation and hardware combinations used. Figure 5.7 illustrates the average runtime of the assignment phase of SATALL, the portion of the application which has been accelerated on the GPU. A subplot is used for each road network benchmarked, with each series corresponding to a separate hardware configuration. In the charts within this section, solid bars are used for the existing serial and multi-core CPU implementations, using colour to differentiate between specific CPU configurations. For GPU results, hatching is used to indicate the number of GPUs, and colour indicates the model of GPU. For instance, system V-2 is indicated by the pink bar with white circular hatching.

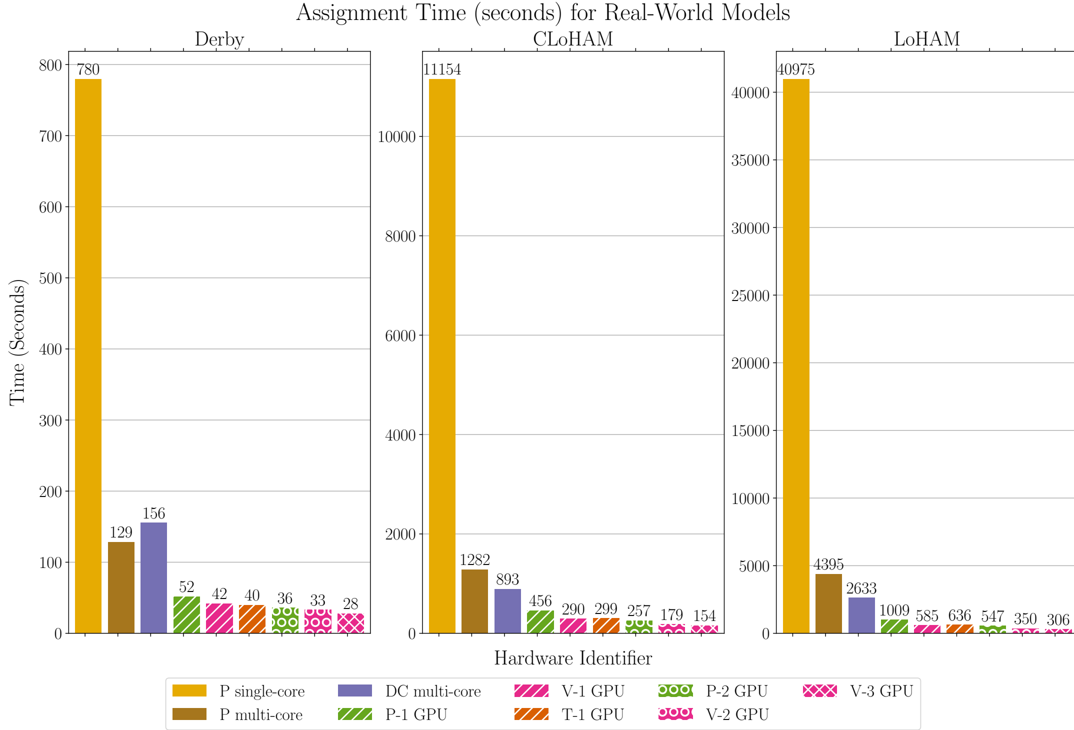


Figure 5.7: The runtime (seconds) of the assignment phase of SATALL for the three real world models on various CPUs and GPUs. Lower times are better. Note that the Y axis is not shared between models.

The GPU implementation which uses the MSSP algorithm shows reduced total processing times and reduced assignment times across all three benchmark models, with greater reductions in runtime for the larger real-world networks. The assignment runtime of SATALL using a single GPU is up to 70 times quicker than the Serial CPU implementation, and up to 7.5 times quicker than the fastest single-socket CPU benchmarks. When using multiple GPUs, a 3-GPU system shows up to 14 times the performance of a single-CPU, and up to 8.6 times the performance of a dual-socket CPU system. These improvements correspond to a reduction in the assignment runtime for the largest network from over 11 hours using a single CPU core or 44 minutes using multiple CPUs down to 10 minutes using a single GPU, or 5 minutes using multiple GPUs.

The smallest network, Derby, shows the smallest improvement from GPU implementations, although it is still an improvement over all CPU-based results. The Derby network contains fewer zones, vertices and edges than the larger CLoHAM and LoHAM models.

This scale of this network limits the performance improvement from the many-core GPU architecture. Within the MSSP algorithm, the Origin-Vertex Frontier does not grow large enough to offer the high degree of parallelism required to extract the most out of the highly parallel hardware, reducing the performance impact of the GPU. For this network, the Titan RTX GPU offers the lowest single-GPU assignment runtime, of 39.8 seconds. This is slightly faster than the Titan V GPU at 41.8 seconds, but both faster than the 51.8 seconds of the Titan Xp in System `P-1`. All three models of GPU used are from the Titan prosumer GPU range. Each

new generation brings architectural improvements, along with higher clock frequencies, more global memory bandwidth, and often larger GPUs. The Pascal architecture Titan Xp of System `P-1` is the oldest of the three architectures. It is also the GPU with the fewest processing cores and streaming multiprocessors, lowest compute performance, and lowest memory bandwidth. These factors all lead to it being the slowest of the three GPUs.

The scale of parallelism within the GPU is an important factor. During periods of the MSSP algorithm where the OVF is largest, the device will be over-subscribed. This is not necessarily a bad thing, as future GPUs which typically contain more processing cores are not under-utilised. However, if there are more threads requested than can be resident within SMs on the device, the blocks of threads are serialised. This can lead to longer runtimes compared to larger devices which can concurrently accommodate all threads. The Titan V GPU can support $163,840$ resident threads over 80 SMs, while the Titan RTX can support $73,728$ threads across its 72 SMs, compared to only 30 SMs and $61,440$ maximum resident threads. The newer architectures also offer higher levels of compute performance per SM, and higher global memory bandwidth. The performance difference between the Titan RTX and Titan V GPUs for the Derby network is most likely due to the increased processing performance of the Turing architecture, and the marginally higher global memory bandwidth.

For the middle-scale CLoHAM network, the Titan V outperforms the Titan RTX GPU, unlike in the Derby network. In this case, the scale of the OVF and total trip matrix used within the flow accumulation phase are sufficiently large that the additional parallelism within the Volta generation card is beneficial. The gap in performance between the Volta and Turing GPUs in systems `V-1` and `T-1` has also grown compared to the Pascal-generation GPU in `P-1`. This pattern also applies to the larger LoHAM network, for the same reasons.

Upon further inspection of the performance of the MSSP algorithm for each network, the CLoHAM network on average requires a relatively large number of iterations relative to the scale of the network compared to than the larger LoHAM network. There are a relatively large number of iterations where the OVF does not contain sufficiently many origin-vertex pairs to fully utilise the device, suggesting the the network has a relatively high diameter, or that there are chains of links which are sequentially updated. A denser representation of the CLoHAM network would likely improve this, which could be achieved through additional, more aggressive pre-processing during the application of contraction hierarchies to create the Spider representation of the graph. Denser representations would likely also improve the performance of shortest path calculations for other algorithms, however, a trade-off is made between the improved shortest path calculation performance, and the additional costs of creating, storing and conversion to and from the denser representation.

In addition to the reduced assignment runtime for the GPU implementation, the remaining part of the total simulation time is also reduced compared to the CPU implementations. For

the LoHAM network, 597 seconds of the 4992 second runtime occur outside of the assignment portion of the application, compared to only 319 of the 903 seconds using a single Titan V GPU in system `V-1`. This is due to different compilers being used for the alternate implementations, with the Sivlerfrost compiler being used for the majority of the CPU implementations, while the PGI Fortran compiler is used for the GPU results for improved GPU support. The PGI compiler is more successful at optimising the CPU-based portions of the application, although no source changes were made.

In Section 5.2 it was shown that the majority of time spent in the serial implementation of SATALL is during the assignment phase of the application. The proportion of time spent in the parallelised assignment portion of the application can give insight into the impact of the optimisations made through algorithmic changes. For the serial implementation, assignment runtimes account for between 65% and 99% of the total application runtime. The existing multi-core implementation brings this down to between 43% and 77% for the multi-CPU results. Single GPU results bring the assignment portion as low as 33% to 65%, while 3 GPUs bring the proportion to between 25% and 49%. Additional GPUs would offer further improvements for sufficiently large models. The reduction to between 25% and 49% of total runtime suggests that further improvements to the assignment process would have a limited impact on total runtime, and that other portions of the application such as simulation should be targets of future optimisation when following Ahmdal's law [260].

The performance improvements offered by the GPU accelerated implementation using the novel MSSP algorithm can be evaluated by comparing against the reference CPU implementations. However, when comparing benchmarks from different hardware configurations it is important to make fair evaluations. The serial CPU implementation provides a common baseline to which the parallel CPU and GPU implementations can be compared, presented in Section 5.8.1. Comparing a many-core GPU implementation against a single CPU core is not the most fair comparison. Instead, Section 5.8.2 presents the speedup of the single-GPU implementations compared against the single-socket CPU system, to evaluate the performance of a single processor. Multiple-GPU results results are then compared against multi-socket CPU results, providing a fair node-level comparison in section 5.8.3. Lastly, insight can be gained by comparing the performance of the multi-GPU results against the results from a single GPU, which is provided in section 5.8.4.

It is also important to be aware of the cost and power requirements of the hardware used when comparing the performance of CPUs and GPUs. Although GPUs are typically more power-efficient then CPUs for compute performance, they often have a higher Thermal Design Power (TDP) than CPUs. For instance, the Nvidia Titan Xp, Nvidia Titan V and Nvidia Titan RTX GPUs used for these benchmarks all have a TDP of 250W, compared to TDPs of 130W and 140W respectively for the Intel Xeon E5-2667 and Intel Core i7-6850K respectively,

Hardware	Implementation	Total Speed-up			Assignment Speed-up		
		Derby	**CLoHAM**	**LoHAM**	**Derby**	**CLoHAM**	**LoHAM**
P	Multi-core (6)	4.17	7.41	8.33	6.07	8.70	9.32
DC	Multi-core (24)	3.33	9.29	12.14	5.01	12.49	15.56
P-1	1 GPU	8.76	18.88	31.33	15.07	24.47	40.62
V-1	1 GPU	9.71	25.98	46.06	18.64	38.49	70.10
T-1	1 GPU	10.06	25.49	43.61	19.61	37.34	64.46
P-2	2 GPUs	9.95	28.18	48.09	21.88	43.48	74.90
V-2	2 GPUs	10.39	34.80	62.33	23.74	62.45	117.15
V-3	3 GPUs	10.92	37.59	66.65	28.19	72.34	133.86

Table 5.6: Relative speed-up compared to the single-CPU single-core results for single-socket CPU and single-GPU. Note: Serial results are from a single run rather than an average due to runtime.

although actual consumption during use will vary. The upfront cost of the GPUs and CPUs used for these benchmarks should also be considered, with the GPUs having original Manufacturer's Suggested Retail Price (MSRP) of 1200 United States Dollar (USD), 2999 USD and 2499 USD each at release, compared to 1552 USD and 617 USD for each CPUs at release. The actual price at the time of purchase may vary significantly however.

5.8.1 Performance Compared to the Serial Baseline

The serial CPU results from system `P` provide a common baseline to compare the results from parallel implementations, although this is not an ideal comparison. Table 5.6 shows the relative speed-up for the total and assignment portion of the application relative to the baseline serial implementation. Figure 5.8 illustrates the relative speed-up of the assignment portion of SATALL for single-CPU and single-GPU systems relative to the serial baseline.

The existing multi-core CPU implementation provides improvements over sequential implementation as expected. Assignment speedup of up to a factor of 9 are shown using the 6 core, 12 thread CPU in system `P`, and up to a factor of 15 for the multi-socket dual-CPU system `DC` which has 24 CPU cores in total. However, for the smaller Derby model the single socket CPU outperforms the dual-socket CPU. The Derby model is too small to fully leverage the additional parallelism of the 18 extra cores, which combined with higher single-core performance of the 6-core CPU leads to the reduced performance. The super linear scaling within system `P` is in part due to a change in compiler between the serial implementation and the multi-core implementation, with the parallel portions of the application being compiled with the Intel compiler rather than the Salford compiler.

The single-GPU results show assignment improvements across all three networks, of up to 70x for the LoHAM model using system `V` with the Titan V GPU.

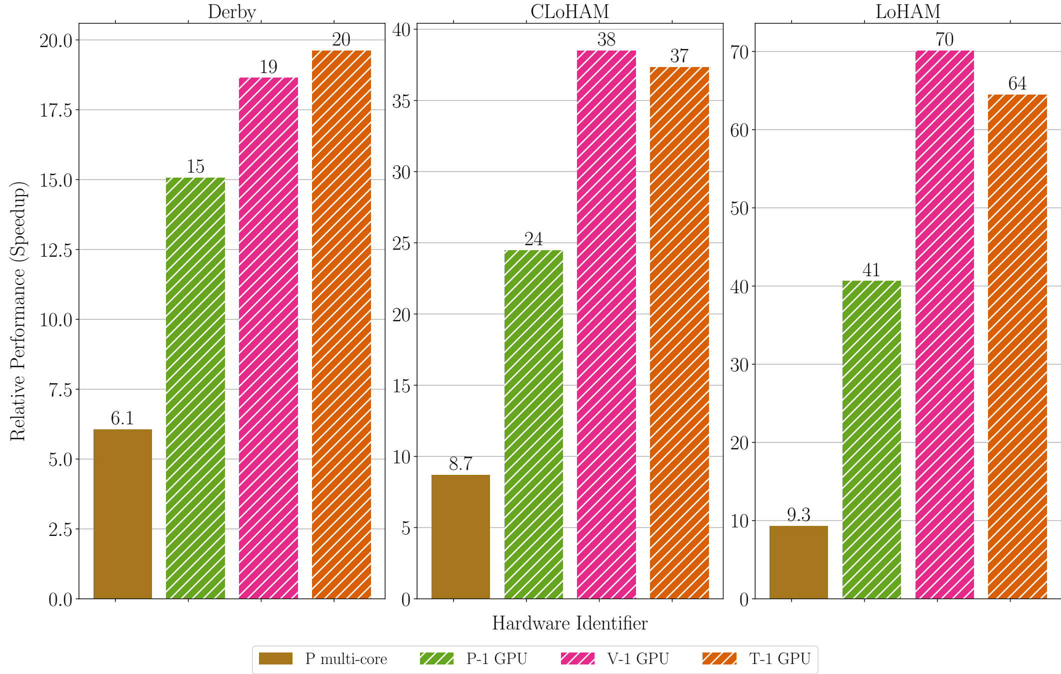


Figure 5.8: The relative speed-up of the assignment phase of SATALL for three real world models compared to serial, single CPU core results. Higher is better.

Hardware	Implementation	Total Speed-up			Assignment Speed-up		
		Derby	**CLoHAM**	**LoHAM**	**Derby**	**CLoHAM**	**LoHAM**
DC	Multi-core (24)	0.80	1.25	1.46	0.83	1.44	1.67
P-1	1 GPU	2.10	2.55	3.76	2.48	2.81	4.36
V-1	1 GPU	2.33	3.51	5.53	3.07	4.43	7.52
T-1	1 GPU	2.41	3.44	5.23	3.23	4.29	6.91
P-2	2 GPUs	2.38	3.80	5.77	3.61	5.00	8.03
V-2	2 GPUs	2.49	4.70	7.48	3.91	7.18	12.57
V-3	3 GPUs	2.62	5.07	8.00	4.65	8.32	14.36

Table 5.7: Relative speed-up compared to the single-socket multi-core CPU results (system `P`). Higher is better.

5.8.2 Single-Processor Performance

Comparing whole processors against one another presents a more balanced picture than comparing against a single CPU core. The relative assignment performance compared against the single-socket multi-core CPU results of system `P` are shown in Table 5.7, although multi-CPU and multi-GPU values are included for reference. Figure 5.9 visualises the relative performance for single-GPU implementations compared to the single-socket CPU results.

The single GPU results show performance speedups from a factor of 2.5 up to a factor of 7.5 for the various networks and hardware architectures. The smaller networks of Derby and CLoHAM show reduced speedups compared to the larger LoHAM model for single GPU implementations compared to a single CPU socket.

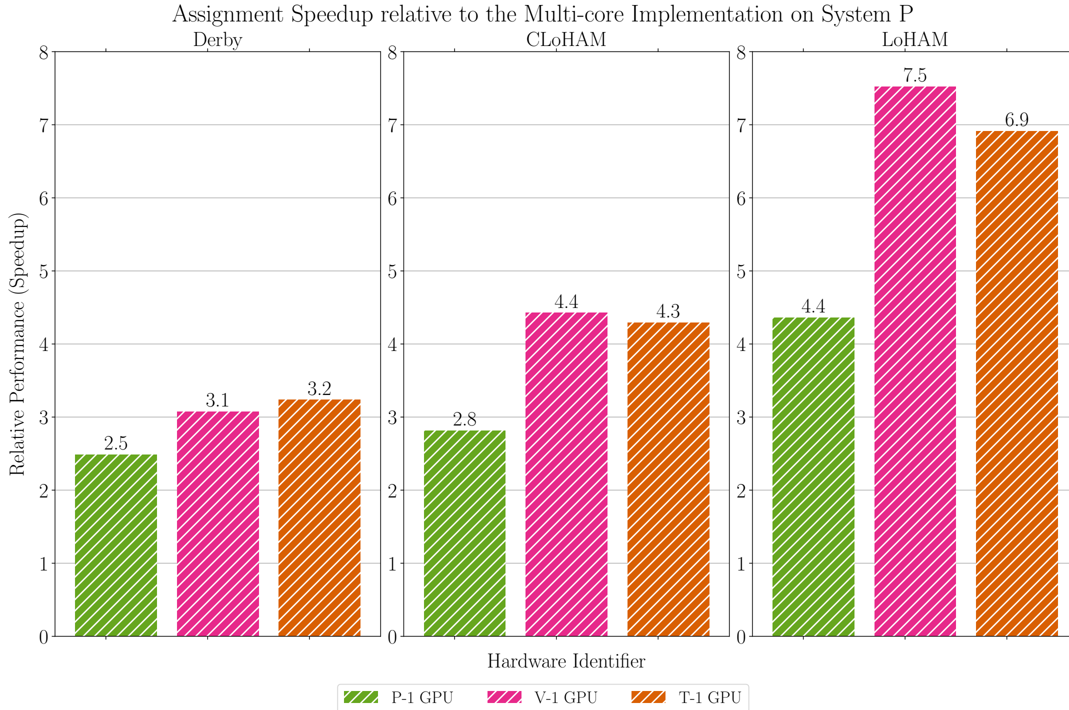


Figure 5.9: The relative speed-up of the assignment phase of SATALL for three real world models on single CPU and single-GPU systems compared to a single multi-core CPU.

5.8.3 Multi-Processor Performance

Computer workstations or servers may contain multiple CPUs and multiple GPUs. When comparing performance results of multi-GPU systems, comparing to multiple CPUs presents a more balanced picture than comparing to only a single CPU. Table 5.8 contains the relative assignment speedup for single-GPU and multi-GPU implementations compared to the multi-core implementation using multiple CPUs. Single-GPU values are included for reference, although the main point of comparison is with the multi-GPU implementation. Figure 5.10 visualises the assignment runtime of multi-socket and multi GPU implementations, and Figure 5.11 shows the relative performance of each multi-GPU implementation against the multi-socket multi-core results.

Single-GPU implementations are only 3x to 4.5x faster than the multi-CPU results, whilst multi-GPU results show relative speedups of between 4.4x and 8.6x. For the LoHAM model, using 3 Titan V GPUs the 8.6x improvement to assignment only results in a total SATALL speed-up of 5.5x, as the assignment runtime for even the largest model is now below 50% of the total application.

5.8.4 Multi-GPU Scaling

Finally, Table 5.9 and figure 5.12 show the relative performance for simulations using multiple GPUs, compared to the single-GPU equivalent, i.e. the runtime of a simulation executed on

Hardware	Implementation	Total Speed-up			Assignment Speed-up		
		Derby	**CLoHAM**	**LoHAM**	**Derby**	**CLoHAM**	**LoHAM**
P-1	1 GPU	2.6	2.0	2.6	3.0	2.0	2.6
V-1	1 GPU	2.9	2.8	3.8	3.7	3.1	4.5
T-1	1 GPU	3.0	2.7	3.6	3.9	3.0	4.1
P-2	2 GPUs	3.0	3.0	4.0	4.4	3.5	4.8
V-2	2 GPUs	3.1	3.7	5.1	4.7	5.0	7.5
V-3	3 GPUs	3.3	4.0	5.5	5.6	5.8	8.6

Table 5.8: Relative speed-up compared to multi-socket CPU (DC)

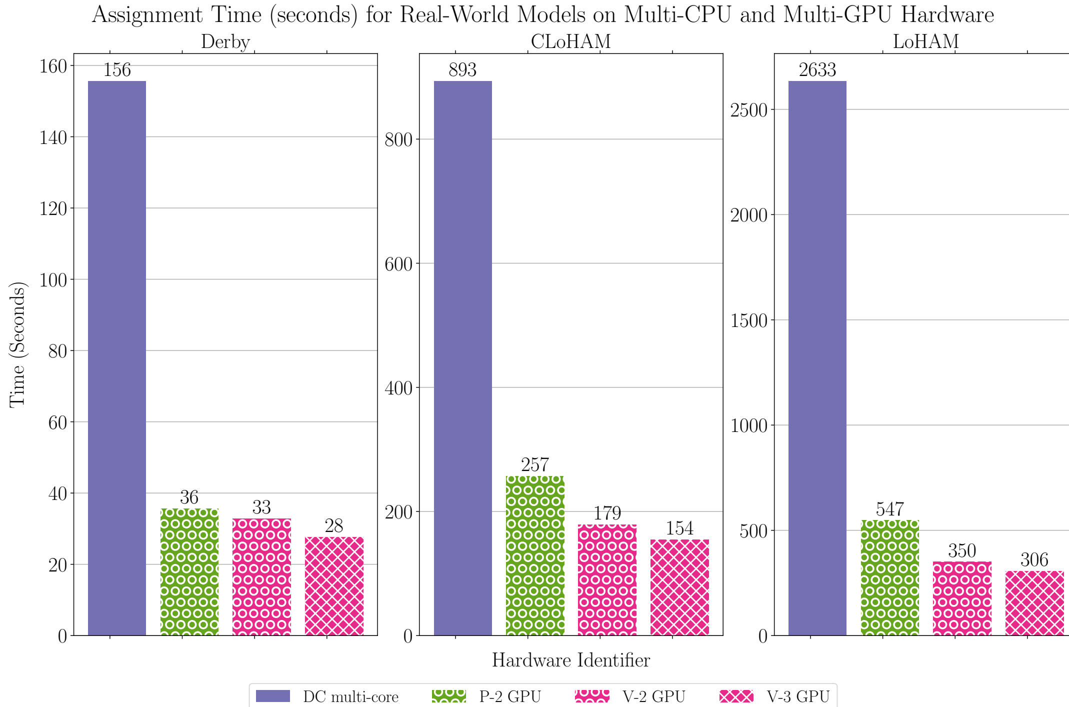


Figure 5.10: The runtime of the assignment phase of SATALL for three real world models comparing multi-socket CPU and multi-GPU implementations.

two Titan V GPUs in system `V-2` is relative to the performance of a single Titan V GPU in the identified as `V-1`.

This shows the strong-scaling behaviour from using additional processor units to solve the same tasks. Diminishing returns are shown from increasing the number of GPUs within the system. Using 2 Titan V GPUs only results in a 67% improvement of performance, and increasing to 3 GPUs only provides a performance improvement of 91% compared to a single GPU (where 100% and 200% would be ideal). This is due to an imbalance of work-load between the devices in the multi-GPU implementation. As discussed in section 5.6, the multi-GPU implementation distributes whole user-classes of work to each device. Table 5.2 shows that the CloHAM and LoHAM networks both have 5 user-classes, resulting in one GPU having to process an extra user-class in each case. The best multi-GPU scaling is achieved when the number of user-classes is a multiple of the number of GPUs. In the case of CLoHAM and LoHAM, there would likely be a large improvement when 5 GPUs are used compared to between 1 and 4 GPUs. Splitting

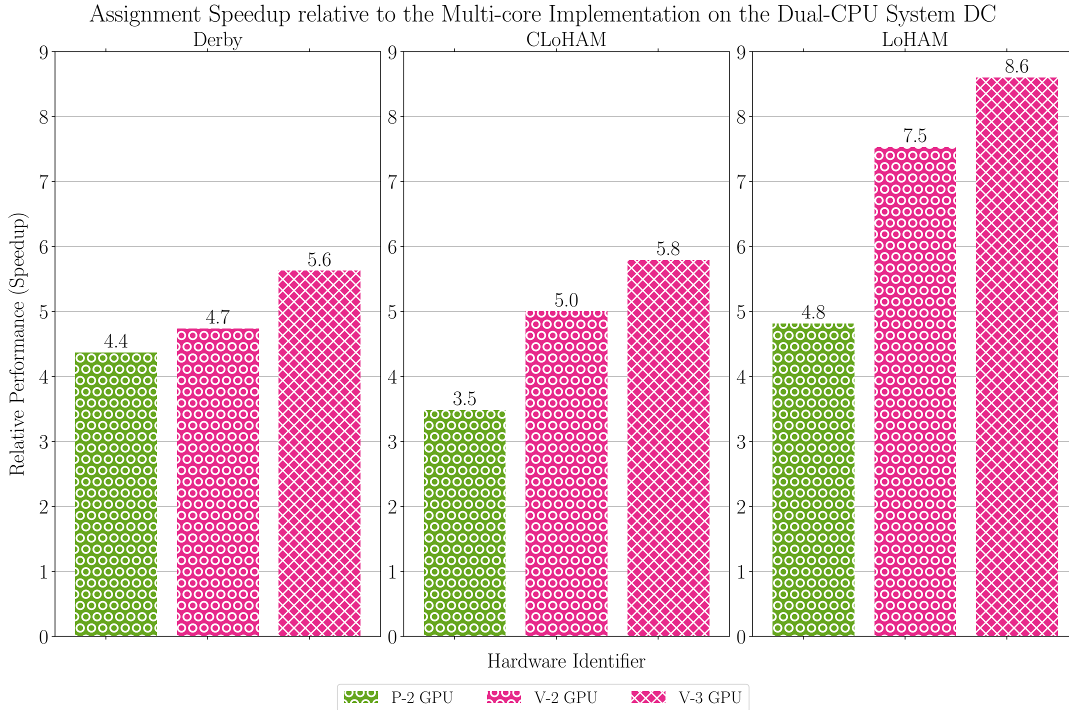


Figure 5.11: The relative speed-up of the assignment phase of SATALL for three real world models on multi-GPU systems, compared to multi-socket CPU results.

Hardware	Implementation	Total Speed-up			Assignment Speed-up		
		Derby	**CLoHAM**	**LoHAM**	**Derby**	**CLoHAM**	**LoHAM**
P-2	2 GPUs	1.14	1.49	1.53	1.45	1.78	1.84
V-2	2 GPUs	1.07	1.34	1.35	1.27	1.62	1.67
V-3	3 GPUs	1.13	1.45	1.45	1.51	1.88	1.91

Table 5.9: Relative speed-up compared to equivalent single-GPU results.

the imbalanced user-class(es) across devices was attempted, but the additional device-to-device memory transfers and synchronisation required outweighed the benefit of doing so on PCI-e based systems. The Derby network does not see as significant an improvement in runtime as splitting the work load across multiple devices results in under utilisation of each device.

The Pascal generation GPUs in system `P` (NVIDIA Titan Xp) show greater improvement from a second GPU than the Volta-generation of GPUs. The Titan Xp GPUs contain fewer streaming multiprocessors and processing cores than the Titan V GPUs, with a maximum of $61,440$ resident threads compared to $163,840$ for the volta generation GPU. This leads to additional serialisation within a single Titan Xp GPU than a single Titan V GPU. However, when using multiple GPUs and the workload is distributed between them, there is less serialisation during portions of the MSSP where origin-vertex frontiers are large enough to over-subscribe the device.

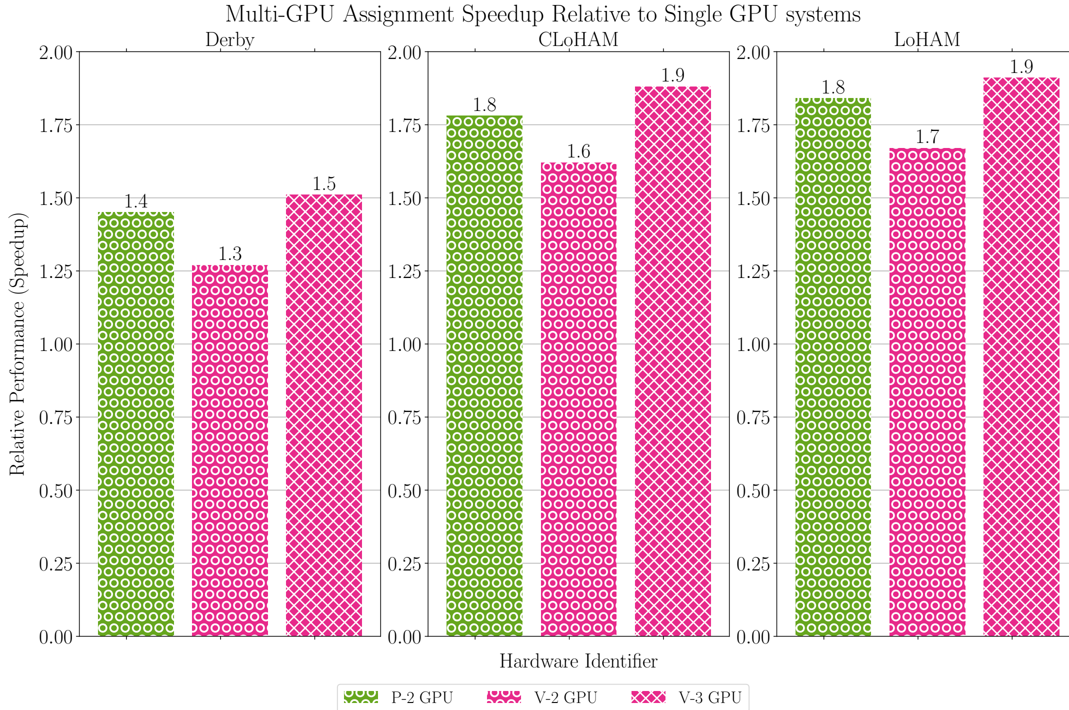


Figure 5.12: The relative speed-up of the assignment phase of SATALL for three real world models on multi-GPU systems, compared to the matching single-GPU system

5.9 Summary

This chapter presented the challenges of applying the GPGPU techniques onto macroscopic road network modelling and simulation. Macroscopic modelling tools such as SATURN often use an iterative loop of demand assignment and network simulation following Wardrop's equilibrium in order to accurately model transport networks. Within the iterative process, a significant portion of time can be spent performing shortest path calculations during the assignment phase of each iteration. The work-efficient shortest path algorithms such as Dijkstra's algorithm which are typically used to achieve high performance shortest path calculations in multi-core CPU environments are not well-suited to many-core processors such as GPUs. Additionally, existing work on GPU acceleration for shortest path calculations, such as the algorithms implemented within Gunrock [226] or nvGraph [229], are focussed on dense-graphs such as social media networks. These algorithms do not perform well when applied to sparse transport networks. A new approach was presented in this chapter to better leverage data-parallelism for sparse graph representations, when many shortest paths must be found.

More specifically this chapter presented a novel Many-Source-Shortest-Path algorithm for many-core processors such as GPUs, expanding on the Bellman-Ford SSSP algorithm. This algorithm enables high performance shortest path calculations on the GPU for sparse networks typical of road networks by solving for many source vertices concurrently, but not all vertices to avoid the demanding memory requirements of APSP algorithms. The data-parallel algorithm

relies on the use of an Origin-Vertex Frontier to efficiently find the shortest paths through the network for many origins, using cooperation between the threads within the SIMT architecture to balance the workload between individual threads to avoid the performance impact of branching. This provides Contribution C5 of this thesis.

Subsequently, the performance of the MSSP algorithm is evaluated through implementation within SATURN, a commercial tool for macroscopic road network assignment and simulation, and benchmarking on a set of real-world transport networks used within industry in the UK, as Contribution C6. This also involves a data-parallel approach to make use of the shortest path results, to avoid costly memory transfers between the device and the host. A single GPU implementation demonstrated assignment improvements of up to 65x compared to a serial CPU implementation, 7.5x compared to a single-CPU multi-core implementation and up to 4.5x compared to a dual-CPU multi-core implementation. Using multiple GPUs, performance improvements of up to 134x were demonstrated compared to serial CPU results, 14.4x compared to single-CPU multi-core results and up to 8.6x compared to dual-CPU multi-core results. The multi-GPU implementation demonstrates relatively poor scaling for the networks in use, due to an imbalanced work-load assigned to each device when the number of devices does not equally divide the number of user-classes within the model. Comparing the existing serial CPU implementation against a highly-parallel single-GPU or multi-GPU implementation is not a particularly fair comparison, with the comparisons of multi-core single-CPU against single-GPU implementations and multi-core multi-CPU implementation against multi-GPU implementations providing fairer comparisons. It is included to provide a common baseline to which parallel CPU and GPU implementations can be compared.

These improvements within the context of SATALL have reduced the proportion of time spent in the assignment phase of SATALL from 99% for the serial implementation down to 49% for the largest model. As such, from the perspective of SATALL total runtimes, the simulation portion of SATALL is the next candidate for further optimisation, as greater improvements to the assignment phase will yield smaller and smaller improvements on the total application runtime. However, there would still be scope to improve the performance of the assignment phase. Creating denser representations of the transport network through modification to the contraction-hierarchy process would likely yield a performance improvement, due to an increase in parallelism exposed by the model, resulting in less time under utilising the GPU during shortest path calculation.

This chapter has demonstrated that many core processing architectures can provide significant performance improvements for large-scale macroscopic road network assignment and simulation models. The previous chapters 3 and 4 demonstrated the application of many-core processors to microscopic road network simulations, leading to significant performance improvements. However, the macroscopic and microscopic approaches see differing degrees of

improvement, due to the degree of parallelism available within the modelling paradigm. In order to more fully understand and compare the performance scaling implications of the GPU approach for these two modelling approaches, and how changes in performance may effect model selection, a comparative study between the two approaches is required. This is the topic of the next chapter.

Chapter 6

Comparison of GPU Accelerated Road Network Simulation Approaches

6.1 Introduction

Road networks can be modelled and simulated using different levels of granularity. Chapters 3 and 4 presented techniques for accelerating microscopic road network simulations using GPUs, with a simple model and procedurally generated network suitable for benchmarking road network microsimulation implementations. Chapter 5 presented details of the macroscopic approach, and methods of accelerating the demand assignment phase of the iterative assignment-simulation form of macroscopic road network modelling.

In Chapter 2, the advantages and disadvantages of each modelling resolution were reviewed. Typically, within the road network modelling sector, macroscopic road network simulations were used in favour of the finer-grained mesoscopic or microscopic simulations, with simulation performance playing a key part in the decision [261]. Additionally, the highly stochastic nature of microscopic models requires a large ensemble to be completed, further exacerbating the total processing time required for microscopic simulations compared to the often more deterministic macroscopic simulations [36]. This reduced variance can also lead to simpler analysis for modelling practitioners.

This chapter aims to evaluate the relative impact of GPU acceleration on microscopic road network simulations and macroscopic road network simulations, presented across Chapters 3 to 5, and how this may impact the choice of modelling approach used. This will provide Contribution C7 of this thesis.

Chapters 3 and 4 proposed and benchmarked a procedurally generated, grid based road network designed to benchmark microscopic road network simulations. Chapter 5 used real-

world road networks used within industry in the UK to evaluate the performance impact of GPUs on macroscopic road network simulations. The relative improvements to simulation performance of each approach can be used to compare these, however, a fairer comparison could be made if the same system were to be modelled using each approach.

In this chapter, the microscopic road network simulation benchmark model from Section 3.4 is adapted for use within the macroscopic road simulation application SATURN, accelerated using GPUs within Chapter 5. Section 6.2 describes the adaptions made to the network. Section 6.3 follows the grid-scale experiment of Section 4.5.1 to benchmark the performance of the macroscopic simulation, presenting results for Central Processing Unit (CPU) and Graphics Processing Unit (GPU) based simulations. Section 6.4 uses these new benchmark results to compare the performance of the macroscopic simulator against the microsimulation results from chapter 4, and the relative impact on the modelling approach is discussed. Finally, Section 6.5 summarises the chapter.

6.2 Benchmark Network

Rather than just comparing the relative performance impact of GPUs on microscopic road network simulations presented in Chapters 3 and 4 against the relative improvements for macroscopic road network simulations in Chapter 5, a more direct comparison is made by comparing the performance of each approach using a common benchmark road network. The procedurally generated Manhattan-style grid network used to benchmark the microscopic road network simulations in Section 3.4 is a good candidate for this. The procedurally generated grid allows the network to be used for tiny networks containing as few as 4 sections of road and a single junction up to vast networks which can approximate national scale road networks in terms of capacity. The simplicity of the model also reduced the set of features which must be implemented within the microsimulation.

The artificial road network consists of a 2D Manhattan-style grid made up of single-lane one-way roads, with adjacent rows and columns of the grid network traversing in opposite directions. Each junction has two input sections of road, and two output sections of road. Sections of road around the edge of the grid are used as entrances and exits into and out of the simulated region. For the microscopic model this is where vehicles are created or destroyed, while in the macroscopic model these map to the zones used for the Origin-Destination (OD) demand matrix. Junctions are simple, using stop signs and give-way modelling in place of more complex dynamic infrastructure such as traffic lights.

Figure 6.1 shows the original design of the artificial road network used to benchmark the microscopic road network simulators, including the arrangement of roads, junctions and turns within each junction. For a given *grid size* G, the generated network contains G^2 junctions,

$2G(G+1)$ road sections, $4G^2$ turning sections, $2G$ entrances and $2G$ exits.

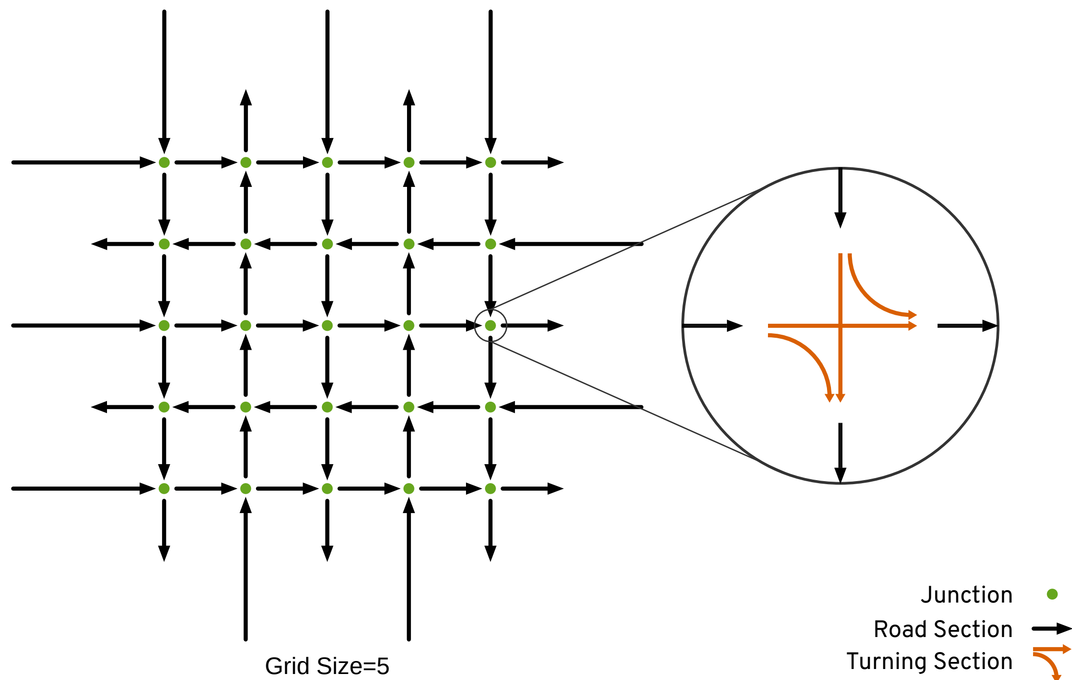


Figure 6.1: A 5×5 example of the procedurally-generated artificial grid network used for microscopic simulations, showing the overall structure of the network and the arrangement of turning sections within a junction. The network can be scaled to any size, with networks of up to 576×576 used during benchmarking.

For the CPU and GPU macroscopic simulation implementations, several key changes had to be made due to the different modelling approach used within SATURN and the types of feature available within SATURN when coding networks. Due to the differing levels of granularity of the modelling approach, there are certain elements of the microsimulation model can not be directly modelled in the macroscopic SATURN simulator. Demand modelling is also approached differently, using demand-based assignment rather then stochastic behaviours per individual. Figure 6.2 shows the adapted version of the network for the macroscopic simulator. The adapted network, for a given *grid size* G, contains $4G$ zones, G^2 regular vertices and $2G(G+1)$ edges, although the denser, Spider representation should contain a larger number of edges and fewer vertices.

Unlike the Aimsun and FLAME GPU networks, SATURN networks include the concept of Zones or Centroids as virtual zones used as entrances and exits into the simulation. Vertices are also classified as either buffer vertices or simulation vertices which determines whether junction simulation will occur or not. The core grid of junctions are all marked as simulation vertices, while entrance vertices are marked as buffer nodes. A Zone is attached to each entrance or exit edge within the simulation (where the edge connects a buffer node to a simulation node) to allow demand to be assigned to the road network, and vehicles to enter and exit the simulation. Figure 6.2 shows the scalable grid network, adapted for SATURN. Buffer vertices are shown

by pink squares, and simulation vertices are shown as green circles. Each buffer vertex has an associated zone connected to the downstream edge, but this is not shown for simplicity. The edges representing road sections are shown by black arrows. A detailed view of the junction turn layout is shown for one of the junctions.

Due to differences between the models implemented in SATURN and the microscopic simulator of chapter 3, the exact junctions modelled in the benchmark network for the microsimulation model could not be replicated exactly. Explicit stop signs can not be encoded in the SATURN network input file. Instead, the sections of road which would have used a stop sign are modelled as give-way turns (SATURN turn priority marker G) [202]. Within the microscopic road network model, all roads are single lane and the fine-grained modelling of individual vehicles within 2D space allows vehicles to avoid one another while traversing junctions. In SATURN however, the junctions had to be implemented using additional lanes at the approach to each junction, to allow the choice of turning actions to be coded into the network. This is achieved using a "flare lane", where a second lane appears just prior to the junction, on the appropriate side of the road for the specific junction. Conceptually, this leads to another modelling difference, where there is potential for vehicles from the same section of road to be alongside one another in the junction.

The benchmark network in the microscopic model used box junctions to impose a restriction on the number of vehicles within a junction at once. The turn geometry was generated such that opposing vehicles intending to make right hand turns would perform nearside turns. Within SATURN, the turn priority modifier X can be used to specify box junctions, however, this models opposing right hand turns (for left-hand drive) as offside turns.

Another difference between the implemented network models is that the there is no concept of vehicle detectors as modelled items of infrastructure, due to the macroscopic nature of the model. Instead SATURN automatically captures metrics about sections of road, which can be accessed through select link analysis procedures.

The microscopic model uses per-individual random number generation to stochastically model the route taken by individuals through the network, using a 50% chance when making a turn between two junctions, or a reduced probability to turn onto a simulation exit. The modelling approach in SATURN instead relies on the iterative process of assignment and simulation to map demand information onto the network. To approximate the per-turn behaviour the OD matrix is initialised to evenly distribute the matching demand for the 1 hour period from the microsimulation model to each destination zone.

The microscopic model was also designed to use per-vehicle properties sampled from random distributions, to model each individual vehicle's properties such as length or acceleration. Within SATURN, vehicle user-classes are used to represent a population of vehicles, where the same parameters are used for all members of that group. Typically this is used in SATURN

to distinguish between classes of vehicle such as Cars, Light Goods Vehicles (LGVs) or Heavy Goods Vehicles (HGVs). For this benchmark network model, 4 user-classes of vehicle are used. The SATURN network files enable the SPIDER SATURN parameter, as this is required to provide a graph representation with enough density to justify the use of GPUs. This should not effect the the modelled behaviour.

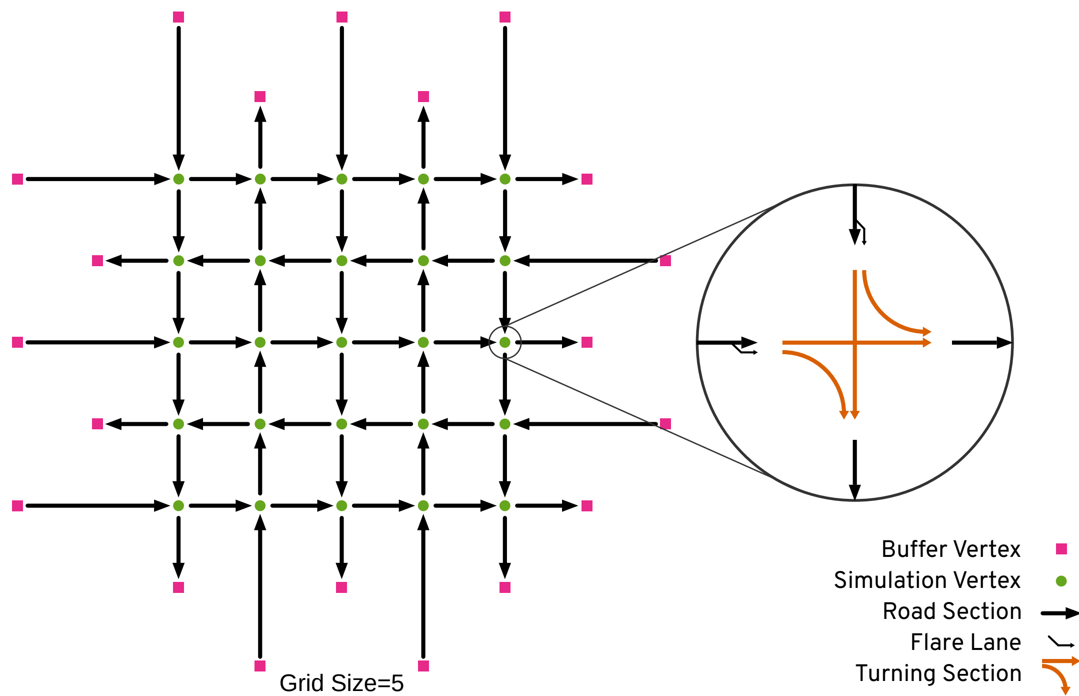


Figure 6.2: A 5×5 example of the procedurally-generated artificial grid network used for macroscopic simulations, showing the overall structure of the network including zones (pink squares), vertices (green circles), road sections (black arrows) and the arrangement of turning sections within a junction (orange arrows) including. The additional "flare lanes" are also shown. The network can be scaled to any size, with networks of up to 122×122 used during benchmarking.

6.3 SATURN Grid Network Benchmarking

By conducting an experiment into total runtime at differing scales it is expected that the results will give us insight into the relative performance benefits of GPUs on macroscopic and microscopic road network simulations, including more-comparable results than just using relative performance impact alone. The Grid-Scale benchmark experiment from Section 4.5.1 is designed to evaluate the performance of a road network simulation at various scales, with the same input demand per entry into the network. This network has been adapted for the macroscopic assignment and simulation tool SATURN (described in Section 6.2), so that the same benchmark experiment can be applied, and the performance used to evaluate the relative impact of GPUs on macroscopic road network simulation. Section 6.3.1 provides the SATURN parameters used for the set of benchmarks simulations. The CPU and GPU versions of SATURN

11.3 from Chapter 5 are executed three times at each scale of the experiment, with the average total, assignment and simulation runtimes detailed in Section 6.3.2. SATURN simulations were executed using the hardware detailed in Table 6.2.

6.3.1 Benchmark Model Parameters

Table 6.1 details the key SATURN parameters used with this benchmark. The numbers of iterations for the assignment and simulation portions within SATURN were selected based on the values used for the real-world networks from chapter 5.

Unlike in the microsimulation benchmark, the *Grid Size* parameter of the procedurally generated network was varied from 4 to 122, rather than between 4 and 576. This upper bound of grid scale is much lower than that used with the microscopic simulation tools due to the use of fixed-size arrays within the *X7* variant of the SATURN software suite. This software limitation is not inherent to the modelling approach, it is merely an implementation detail within the network processing within SATURN. Several work-around options were investigated, but they were dismissed to avoid introducing larger discrepancies between the network being modelled.

Parameter name	**Value**	**Units**
Grid size	4 - 122	
User Classes	4	
Simulation Time	60	minutes
Road Section Length	1000	m
Free Flow Section Speed	50	kph
Capacity Section Speed	12	kph
Link Capacity	2000	PCU/h
Input flow	600	PCU/h
Flow delay curve power	3	
Assignment Iterations (NITA)	60	
Minimum Assignment Iterations (NITA_M)	60	
Minimum Simulation Iterations (NITS)	15	
Minimum Simulation Iterations (NITS_M)	15	

Table 6.1: Parameters used for macroscopic simulations benchmarks of the artificial road network.

6.3.2 Macroscopic Assignment and Simulation Results

Each simulation from the grid-scale experiment was repeated 3 times to provide average runtime values. Simulations were performed on the CPU using both serial and multi-core implementations. GPU simulations were performed using both single-GPU and dual-GPU configurations. Table 6.2 describes the hardware used to benchmark the CPU and GPU versions of SATALL for the grid-scale experiments.

Table 6.3 contains the average runtime values for the SATURN CPU and GPU simulations, including the Total runtime, the runtime of the assignment portion of the application which

ID	CPU Model	CPU Cores	CPU Freq(GHz)	GPU Model	GPU Cores	GPU Freq(GHz)
CPU	i7-6850K	6	3.6			
GPU	i7-6850K	6	3.6	2x Titan V	5120	1.200

Table 6.2: The hardware used to benchmark CPU and GPU SATURN simulations for the grid-scale experiment using the Manhattan-style procedurally generated grid network.

has been accelerated for GPUs, and the serial CPU-based simulation portion of runtimes. Figures 6.3 and 6.4 illustrate the raw runtime values for the total processing time and assignment portion of runtime portions of runtime respectively.

The total runtime results show the expected behaviour of the serial CPU implementation showing the slowest runtimes, followed by the multi-core CPU implementation, and then the single GPU implementation. The multi-GPU results show very similar performance to the single-GPU results at this scale of network. This suggests that the model is either not sufficiently parallel to fully utilise the Titan V GPU, or that the performance is not limited by the degree of parallelism available, but rather the number of iterations of the Many Source Shortest Path (MSSP) algorithm required to find the correct results.

For the serial CPU implementation, the assignment portion of the iterative process only accounts for 37% of the runtime of the largest grid simulated. This suggests that this network is unlikely to see significant improvements from the GPU assignment, as it is already only a small portion of the total runtime. The multi-core, single-GPU and dual-GPU implementations all provide similar assignment runtimes to one another. This is in contrast to the results of Chapter 5 where the GPU implementation was able to reduce the runtime of the simulation for the three benchmark networks, however, for the Epsom network used as a trivial example this was not the case. This suggests that the scales of graph used here do not sufficiently occupy the GPU to justify the cost, as confirmed by the comparable single and dual-GPU results. The specific implementation which offers the highest performance at any given scale does vary slightly across the range of implementations. At grid-scales fewer than 64, the multi-core CPU implementation using 6 cores and 12 threads offers the shortest assignment runtime. In this case, the overhead costs associated with performing the shortest path operations on the GPU, such as data transfer, likely outweigh any performance advantage the MSSP algorithm can offer. Larger grid-scales, up to 112 show assignment advantages for the GPU implementation, although they are relatively minor. At the largest grid sizes of 120 and 122, both simulators re-converge in terms of performance.

One aspect of the road network design which may play a part in this assignment performance is the uniformity of the grid-based road network on assignment. A grid-based layout where edges will have the same initial costs (due to being the same length with the same effective speed limit),

will have many routes with equivalent shortest cost. This may lead to the convergent algorithm flipping between competing shortest routes, for any given origin-destination pair. It could also be that the spider process is short-cutting many of the routes, producing even very short routes which favour work-efficient serial shortest path implementations.

As the assignment of the multi-core and GPU implementations are similar, the total runtime performance difference must come from another portion of the application. Figure 6.3 shows the simulation portion of the runtime for each implementation at each scale of network. This shows that both serial and multi-core CPU implementations show similar simulation runtime to one another. This is expected as the simulation portion of the serial and multi-core SATALL executables is executed in serial, and compiled using the silverfrost fortran compiler. The GPU executable results are approximately twice as fast as the CPU implementations, even though the simulation portion of SATALL is still executed in serial on the GPU. This is due to the use of the PGI compiler, which more aggressively optimises the serial CPU code than the Silverfrost compiler used for the CPU SATALL executables.

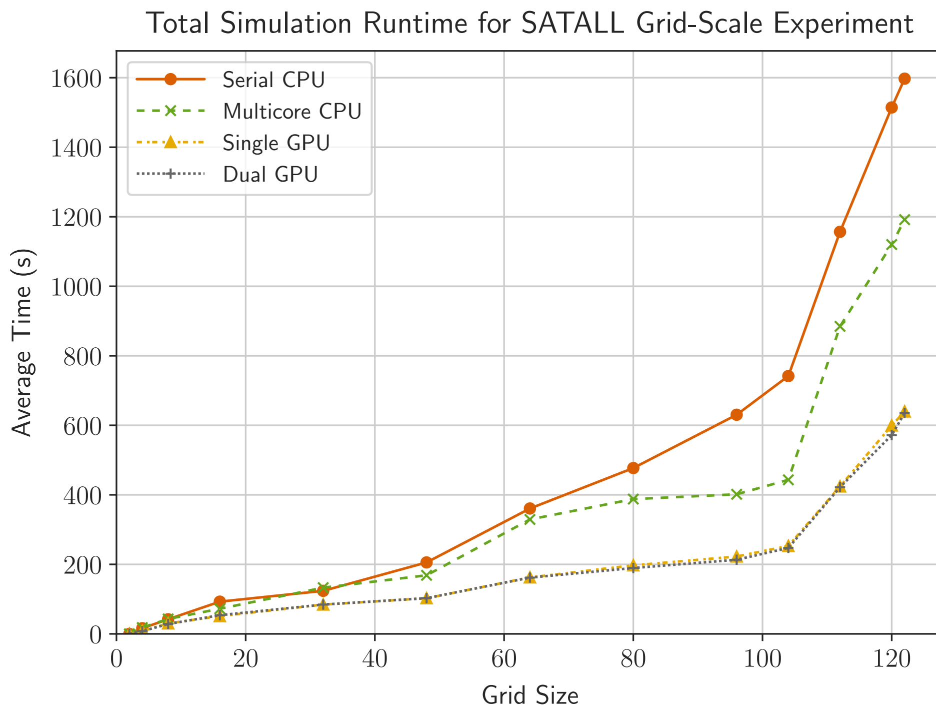


Figure 6.3: The average total runtime of SATURN for the grid-scale benchmark, for CPU and GPU implementations. Values shown are the average from 3 repetitions.

The relative performance improvement of the single GPU implementation against the multi-core CPU implementation is shown by Table 6.4. This shows that total runtime improvements of between 1.4 and 2.2 times the performance of the multi-core CPU implementation was achieved. This is mostly from the simulation speedup of the alternate compiler, which accounted for a significant portion of the total runtime for this road network. The relative assignment performance ranges from 0.1 times that of the multi-core implementation, to up to 1.2x for a grid

Implementation	Gridsize	Total Time (s)	Assignment Time (s)	Simulation Time (s)
Serial CPU	2	0.04	0.00	0.01
Serial CPU	4	16.25	0.07	16.14
Serial CPU	8	42.04	0.39	41.60
Serial CPU	16	92.74	1.98	90.67
Serial CPU	32	123.80	11.40	112.16
Serial CPU	48	205.55	34.91	170.15
Serial CPU	64	360.69	82.67	277.08
Serial CPU	80	477.07	161.75	313.71
Serial CPU	96	630.40	270.06	357.80
Serial CPU	104	741.72	348.89	389.66
Serial CPU	112	1156.65	449.64	702.98
Serial CPU	120	1514.37	541.50	968.12
Serial CPU	122	1597.22	589.60	1002.63
Multi-core CPU	2	0.17	0.15	0.01
Multi-core CPU	4	18.20	0.52	17.63
Multi-core CPU	8	43.03	0.90	42.08
Multi-core CPU	16	72.32	2.35	69.88
Multi-core CPU	32	133.00	7.62	125.14
Multi-core CPU	48	168.54	17.97	150.07
Multi-core CPU	64	329.24	29.28	299.01
Multi-core CPU	80	387.80	48.31	337.86
Multi-core CPU	96	401.55	72.83	326.14
Multi-core CPU	104	443.21	90.30	349.71
Multi-core CPU	112	884.66	109.29	771.27
Multi-core CPU	120	1120.06	130.12	985.14
Multi-core CPU	122	1191.69	137.53	1049.12
Single GPU	2	0.33	0.30	0.00
Single GPU	4	8.10	1.81	6.25
Single GPU	8	30.09	8.42	21.64
Single GPU	16	51.06	12.16	38.79
Single GPU	32	84.29	26.15	57.86
Single GPU	48	102.49	28.52	73.41
Single GPU	64	162.46	28.98	132.36
Single GPU	80	197.17	41.85	153.51
Single GPU	96	222.56	65.15	154.60
Single GPU	104	252.91	75.06	174.43
Single GPU	112	424.06	93.25	326.48
Single GPU	120	599.50	153.57	440.90
Single GPU	122	639.74	149.70	484.72
Dual GPU	2	0.40	0.37	0.00
Dual GPU	4	7.77	1.62	6.10
Dual GPU	8	28.45	7.14	21.27
Dual GPU	16	53.75	14.59	39.05
Dual GPU	32	84.39	26.50	57.62
Dual GPU	48	102.75	28.55	73.62
Dual GPU	64	161.80	28.75	131.9
Dual GPU	80	189.47	35.16	152.48
Dual GPU	96	213.25	56.30	154.21
Dual GPU	104	247.29	68.10	175.79
Dual GPU	112	422.07	91.23	326.51
Dual GPU	120	571.76	125.94	440.81
Dual GPU	122	635.53	145.58	484.72

Table 6.3: Average runtime (seconds) for the total runtime, assignment runtime and simulation runtime of each grid scale benchmark for the serial CPU (i7 6850k), multi-core CPU (i7 6850k using 12 threads), single GPU and dual GPUs (NVIDIA Titan V) macroscopic road network simulation implementation (SATURN).

scale of 104.

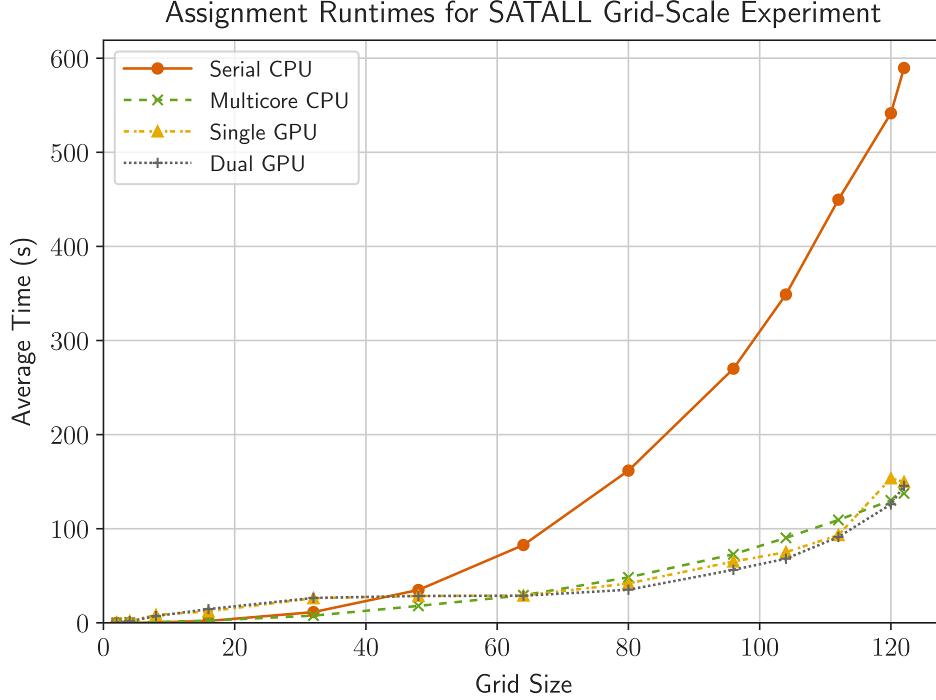


Figure 6.4: The average runtimes of the assignment portion of SATURN for the grid-scale benchmark, for CPU and GPU implementations. The simulation portion is always executed in serial on the CPU. Values shown are the average from 3 repetitions.

Gridsize	Total Speedup	Assignment Speedup	Simulation Speedup
4	2.25	0.29	2.82
8	1.43	0.11	1.94
16	1.42	0.19	1.80
32	1.58	0.29	2.16
48	1.64	0.63	2.04
64	2.03	1.01	2.26
80	1.97	1.15	2.20
96	1.80	1.12	2.11
104	1.75	1.20	2.00
112	2.09	1.17	2.36
120	1.87	0.85	2.23
122	1.86	0.92	2.16

Table 6.4: Relative performance improvement comparing simulator performance for a single Titan V GPU against a multi-core CPU implementation executed on an i7-6850k, for each timed phase of SATURN for the grid-scale macroscopic experiment

6.4 Performance Comparison of Microscopic and Macroscopic Simulations

To evaluate the relative performance impact of many-core parallel processing on the model resolution decision making process, the performance of microscopic and macroscopic road network simulations can be compared for the same transport networks, using the microscopic benchmark results from Chapter 4 and the macroscopic results from Section 6.3.2. However, as these macroscopic results are somewhat restricted in scale, we can gain some insight into the poten-

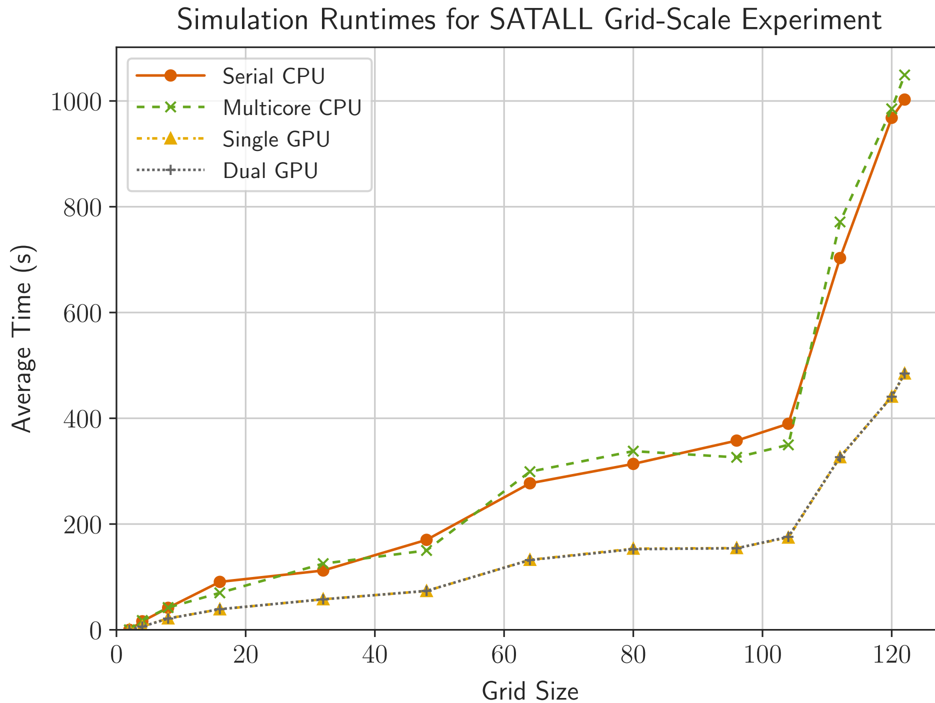


Figure 6.5: The average runtimes of the simulation portion of SATURN for the grid-scale benchmark, for CPU and GPU implementations. The simulation portion is always executed in serial on the CPU. Values shown are the average from 3 repetitions.

tial evaluation of the simulation performance, based on the real-world networks benchmarked in Chapter 5.

The modelling scale decision making process is guided by several factors. In some cases, the choice is dictated by institutional guidance from governmental or professional bodies, which should be periodically reviewed. In other cases, the task may dictate the specific modelling approach required. For instance, fine-grained microscopic or mesoscopic modelling may be required when optimising traffic signal timing or coordination, or when simulating the potential impacts of connected and autonomous vehicles on the transport network. Many scenarios could be successful using any of the three modelling granularities, at which point factors such as the performance, determinism and data requirements may need to be considered.

Macroscopic road network simulations are traditionally less computationally expensive than finer grained modelling approaches, leading to wider adoption, especially for large-scale regional or national models. In contrast, microscopic simulations are typically only used for local, small scale simulations, due to the high computational cost of each simulation. On top of this, the requirement to run many simulations as part of an ensemble for the highly stochastic microscopic simulations also shifts the decision making process towards the more deterministic macroscopic models.

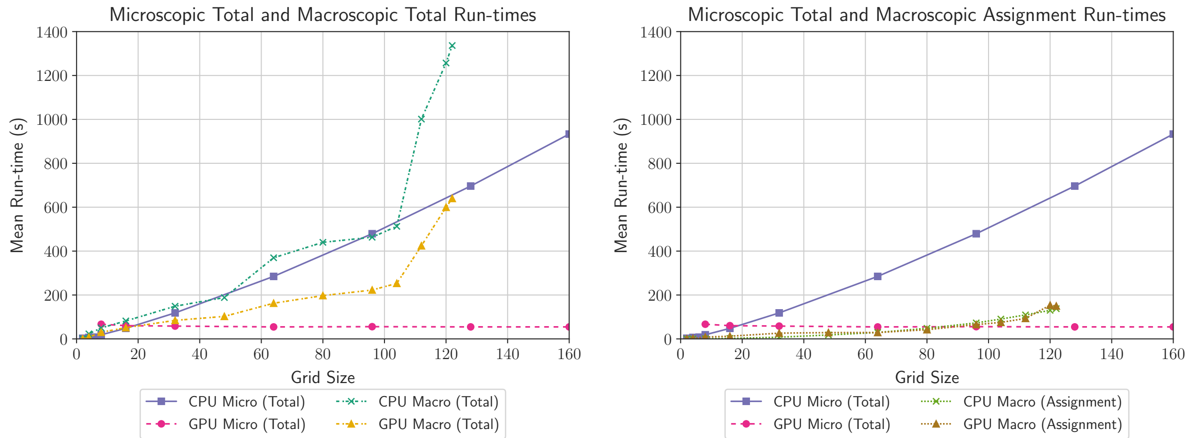
As GPUs can lead to performance improvements for both simulation approaches, the performance of GPU simulations at the same scale can play an important factor in systems which

require a low simulation latency, such as incident response.

We can gain insight into this by directly comparing the results of Section 6.3.2 against the matching benchmarks from Chapter 4. Figure 6.6 presents these results for multi-core CPU applications, and many-core single-GPU systems, with Figure 6.6a presenting the total runtimes of both the macroscopic and microscopic applications, and Figure 6.6b presenting the total runtime of the microscopic simulation against the assignment portion of the macroscopic application. The figures shows the total runtimes of each of the 4 CPU and modelling approach combinations, however, as the CPU-dominated simulation portion of the macroscopic simulations, the assignment portion is also included to gain insight into the GPU performance.

The total time for multi-core macroscopic and microscopic approaches is comparable for grid sizes up to 100, at which point the serial simulation portion of the application begins to consume a significant portion of time, leading to longer runtimes. However, as many microscopic simulations are required for meaningful results the CPU-based microsimulation is less-favourable than the macroscopic approach for these scales of network.

The GPU-based microsimulations shows poor simulation runtimes at small scales. Compounded by the need to run ensembles of many individual microsimulations this suggests that GPU implementations of microscopic road network simulations are unlikely to be impact the choice of modelling approach for small-scale simulations. However, the performance scaling behaviour of the agent-based microsimulation is impressive compared to that of CPU microsimulations and GPU or CPU macroscopic simulations, with minimal runtime increases for all grid-sizes presented. This relative improvement between CPU and GPU microsimulation performance does make it appear much more favourable, when compared to the total runtime of the macroscopic assignment and simulation model. However, as the simulation phase of the application is sequential, this comparison is not entirely fair. Assuming that the simulation portion of the application could be parallelised to a reasonable degree, either using a multi-core or many-core approach, at these scales of simulation the macroscopic approach could still be favourable. This is highlighted by the total time of the GPU macrosimulation, which just through a change in compiler has reduced the time required for a single execution to below that of the largest comparable CPU microsimulation. Broadly speaking, deterministic macroscopic simulations will still have a lower time to completion than an ensemble of microsimulations, for the scales of the benchmark network demonstrated by this direct comparison. This may not be the case for larger-scale simulations, considering the performance scaling of the curves. The CPU-based microsimulation scales with a relatively steep gradient, which is compounded by the need for many simulation runs, making it unfavourable for larger-scale simulations. The GPU-based microsimulations show very low increases in simulation runtime as the scale is increased, when efficient communication is used as proposed in Chapter 4. What can be seen of the macrosimulation results, when excluding the simulation portion and only considering the



(a) Total runtime for both microscopic and macroscopic approaches

(b) Total microscopic runtime and the assignment portion of the macroscopic simulators

Figure 6.6: CPU and GPU performance data against grid size for both microscopic and macroscopic simulations, for both CPU and GPU implementations. Figure 6.6a contains the total runtime for both microscopic and macroscopic approaches. Figure 6.6b shows the total microscopic runtime and the assignment portion of the macroscopic simulators, as this is the section of the application which uses the GPU and MSSP algorithm.

assignment phase, shows reasonable performance scaling until grid-sizes of more than 104, at which point there is a clear change in gradient. Due to the limitations of the grid-network design when implemented for the SATURN road network simulation, a fair direct comparison for larger scales can not be made. However, the shape of the assignment phase of the macroscopic simulator when executed on the GPU does suggest that it will perform less-well than the GPU accelerated microscopic simulations as model scale increases.

Instead, the relative performance improvement between the CPU and GPU based macroscopic and microscopic simulations can be compared, both for the available benchmark data, but also using the figures presented in previous chapters, to discuss the improvement for larger scale macrosimulations, and how it may compare to large scale microsimulations. Figure 6.7 show the relative speedup achieved by the single-GPU implementations compared to their relevant multi-core results. The raw speedup is shown in Table 6.4. This shows total macroscopic speedup of up to 2x, with assignment speedup peaking at 1.2x. The microsimulation speedup however grows as the scale of the simulation is increased, demonstrating that individual microsimulations gain significant performance from GPUs when they are of a high enough scale.

As simulations with grid-scales larger than 122 cannot be directly compared, some insight can be gained from comparing the relative speedup of single-GPU simulations by comparing the relative speedup of the artificial benchmark against the real-world macroscopic benchmarks. The grid-based macroscopic model with a gridsize of 122 has 488 zones. This can be used as a proxy for the scale of the network, suggesting it is close in total work-load the Derby model (although fewer user-classes are used, and the number of vertices and edges in the spider representation of the graph may be significantly different). The LoHAM model contains 5194 zones, which would

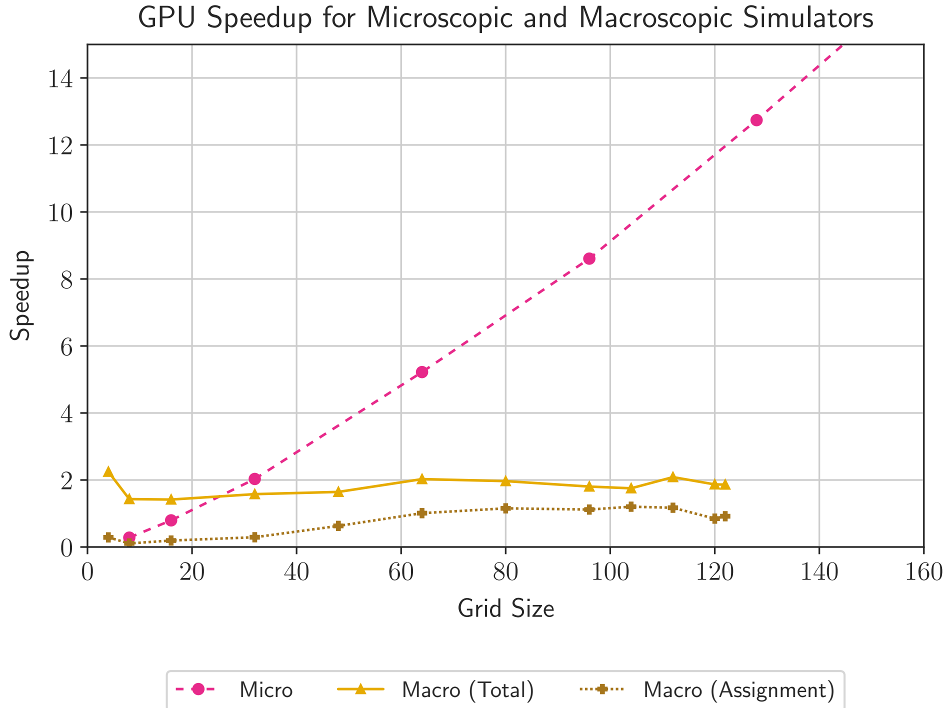


Figure 6.7: GPU implementation speedup relative to the respective CPU implementation for microscopic and macroscopic implementations.

suggest it might be similar to a grid-scale of 1299. CLoHAMs 2547 zones could be approximated as a 636 gridsize model, although these are rough estimates.

The maximum speedup values observed for single-GPU vs single-CPU comparisons in both simulators can be compared, using values from Section 4.5.1 and Section 5.8.3. The GPU-accelerated macroscopic simulator demonstrated relative performance improvement factors of 2.33, 3.51 and 5.53 for the three real-world models compared to the multi-core CPU implementation, in increasing order of size. The assignment phase of the macroscopic simulator, which is the portion accelerated by the GPU, achieved relative performance improvement factors of 3.07, 4.43 and 7.52. These values all suggest that the relative performance improvement increases as the scale of the problem increases. However, these values are still significantly below the speedup of 67x observed for the largest microsimulation benchmark, in Section 4.5.1. This difference in relative performance improvement supports the idea that GPU acceleration reduces the impact of one of the key disadvantages of microscopic road network simulations, especially when compared to the traditionally computationally cheaper macroscopic simulations.

6.5 Summary

In this chapter the microscopic road network simulation results from Chapters 3 and 4 were compared against the macroscopic simulator from Chapter 5. This discussion forms the basis of contribution C7 of this thesis, although there is room for further investigation to be carried

out as future work. The scalable, grid-based benchmark road network was modified to be used within SATALL, to allow a more-direct performance evaluation between the alternate road network modelling methodologies. However, the scale of these benchmarks were limited by limitations within SATALL, and the network in question did not exhibit clear performance improvements for the benchmark road network used, only offering relatively small performance advantages over the existing multi-core implementation. Unlike in Chapter 5, using multiple GPUs did not lead to a clear performance improvement over a single GPU. This also suggests that the scale of network used did not lead to sufficient parallelism within the assignment phase of SATALL.

Due to the limited scale of these benchmarks, the results of the previous chapter are the source of insight into the performance at larger scales. Section 6.4 discussed the potential impact of these relative performance improvements, both directly based on the small scale grid network benchmarks, and with the relative performance improvements shown in previous chapters.

In general, it can be concluded that microscopic road network simulations appear to benefit more from parallelisation through many-core processors such as GPUs. The performance improvements available to microsimulations are clearly demonstrated by Chapters 3 and 4, especially for larger-scale simulations with performance improvements of up to 67x demonstrated when comparing a single-CPU multi-core implementation against a single-GPU implementation. This is enabled by the fine-grained data-parallelism which is inherent to microscopic road network simulations implemented as agent based models, although care must be taken to ensure agents can efficiently communicate with one another in parallel. The real-world macroscopic road network simulation results from the previous chapter do suggest that it is beneficial for larger-scale simulations, but not for smaller simulations which do not expose sufficient parallelism to outperform highly work-efficient CPU based implementations.

Although GPUs can shift the runtime decision making point towards finer-grained simulations, they still present additional challenges compared to higher-level simulations. The highly stochastic nature requires larger ensembles of many simulations for accurate results. This means that the total microsimulation runtime for a full ensemble will likely still consume more time than macro-scale simulations at large scales, but the total time saved by using many-core architectures and data-parallelism can result in considerable performance improvements. Further improvements to GPU hardware will likely increase the level of parallelism available within a single device. Given the higher degree of parallelism exposed by microscopic modelling approaches rather than the macro-scale models, this would suggest that the relative difference improvement in performance between the two approaches will grow as new generations of GPU are produced.

Chapter 7

Conclusions

Models and simulations of transport networks are used globally to analyse, design and manage transport network infrastructure, to achieve goals such as increasing capacity to support ever-increasing levels of demand [1], reducing travel time by improving utilisation of existing transport network infrastructure [2], [3] or increasing safety [262]. Transport models are often categorised by their granularity, ranging from coarse top-down macroscopic approaches, through medium-grained mesoscopic approaches, to fine-grained individual-based microscopic models. Unfortunately, these simulations can be computationally expensive, with long run-times for individual simulations, while many simulations may be required for meaningful results. The performance of simulation tools imposes limits on their effectiveness [14]. Historically, individual computing processors would simply become faster year-on-year, however, that is no longer the case, in-part due to physical limitations on the materials used within semicondutor devices [70]. Instead, processors are containing greater and greater numbers of processing cores, with multi-core Central Processing Units (CPUs) containing tens of complex processor cores, to many-core processor architectures such as Graphics Processing Units (GPUs) which contain thousands of relatively simple processing cores. Road network simulation tools used within the transport modelling sector make good use of multi-core CPUs to reduce the time required for simulations to complete, however, there has been little adoption of many-core architectures which have the potential to offer performance advantages compared to many-core designs. But, to access the performance offered by the highly parallel processing architecture, different algorithms and data structures must be used, potentially increasing complexity and requiring specialist knowledge to achieve.

This thesis set out to answer three questions:

1. Can modern GPUs be used to provide efficient and scalable microscopic road network simulations which offer high levels of performance compared to more traditional CPU-based approaches?
2. Are modern GPUs suitable for the acceleration of macroscopic road network simulations,

to reduce the time required to simulate large-scale road networks?

3. What impact will GPU acceleration have on the choice of modelling approach, between microscopic and macroscopic simulations?

This work has proposed novel algorithms to provide high levels of simulation performance for large-scale simulations of road networks, for both microscopic and macroscopic modelling techniques. Supporting evidence has been provided through experimentation and quantitative benchmarking.

Chapters 3 and 4 sought to address the first of the three questions. In order to evaluate whether many-core GPU processors can provide high performance for microsimulations compared to the current state of the art CPU-based simulators in use today, a set of behavioural models are which must be implemented into a GPU-based simulator. As there transport networks come in many forms, with differing local rules, Chapter 3 first proposed a subset of the models used within Aimsun 8.1 [143], a state of the art commercial microscopic road network simulation tool, which would be sufficient to implement a scalable benchmark model. This subset of models was complemented by the definition of a scalable, procedurally generated artificial road network, suitable for benchmarking this subset of models at any scale. Together these two parts form Contribution C1 of this thesis. Chapter 3 also provides the description of an GPU-accelerated implementation of this microscopic road network model as an Agent Based Model (ABM), using the Flexible Large-scale Agent Modelling Environment for Graphics Processing Unit (FLAME GPU) framework [184]. Several benchmark experiments are performed, using both simulation engines, to evaluate the performance at a range of scales. This highlights the need for efficient communication between agents within the transport network to reduce the cost of communication within the many-core environment and enable high levels of performance for large-scale simulations.

Subsequently, Chapter 4 proposes a general-purpose graph-based communication strategy for GPU accelerated ABMs. The graph-based communication pattern provides a mechanism for agents to efficiently communicate with one another based on locality within a road network, or any other system which could be represented by a graph. It reduces the number of messages each agent must iterate in order to find the relevant data required by the road network behavioural models, resulting in improved work efficiency compared to the other communication techniques available in FLAME GPU [22]. This addresses the sub-question of what algorithms and data structures may be required to accelerate microsimulations using GPUs, and provides contribution C3. A benchmark ABM is proposed to along-side this communication method, to evaluate the performance of the communication strategy in isolation from the complexities of a complete road network simulation model, and forming Contribution C3. This was vital in demonstrating and understanding the performance for the generalised approach. The benchmark ABM, inspired by simulations of molecules of water flowing through a network of pipes

is benchmarked, demonstrating significant performance improvements when compared to alternate communication strategies embedded within FLAME GPU. Performance improvements of several order of magnitude are demonstrated for simulations containing half a million agents.

Chapter 4 continues by applying the novel communication strategy to the microsimulation model and benchmark defined in Chapter 3, and the performance evaluated using a series of benchmark experiments (Contribution C4). For a one-hour simulation of up-to $512,000$ vehicles and $1,575,936$ agent-based pieces of road network infrastructure, the GPU accelerated simulator using the novel graph-based communication strategy achieves a real-time-ratio (RTR) of 44.7, and a relative performance improvement of 67.7x is shown compared to the commercial multi-core CPU microsimulator Aimsun.

The second question, concerning the suitability of GPUs for macroscopic road network assignment and simulation models is answered within Chapter 5. First, the areas of a macroscopic model which would benefit the most from GPU acceleration must be identified. Through application profiling the calculation of shortest paths through the transport network was found to account for a significant proportion of macroscopic runtime. Many shortest paths must be found, to assign transport demand routes through the network, at each iteration of a convergent algorithm. Road network graphs are typically sparse, high-diameter graphs, which do not achieve high levels of performance from state of the art Single Source Shortest Path (SSSP) algorithms when implemented for highly parallel many core processor architectures [225]. Instead, a novel Many Source Shortest Path (MSSP) algorithm is proposed which mitigates the lack of parallelism exposed by sparse road networks. The MSSP algorithm for many-core processors extends vertex-frontier based Bellman-Ford SSSP algorithm [213] to concurrently find the shortest paths from many origin vertices, using an Origin-Vertex Frontier (OVF). Methods to achieve high performance are described, including the use of Cooperative Thread Arrays (CTAs) to load-balance the unbalanced workload caused by varying vertex-degree throughout road network graphs. This algorithm provides contribution C5.

Chapter 5 continues by describing how the MSSP algorithm was embedded The MSSP algorithm is then embedded within a commercial macroscopic road network tool, SATURN [202], to enable benchmarking against real-world transport network models used within the UK, including large-scale regional models [253], [254]. A single-GPU implementation demonstrated performance improvements of up to a factor of 65x compared to the serial version of SATURN, and 7.5x compared to a single-CPU multi-core benchmark. A multi-GPU implementation, which distributes independent units of work across the available devices showed speedups of up to 8.6x compared to a dual-CPU multi-core benchmark for a region-scale macroscopic model. This performance evaluation demonstrates the potential impact of the GPU-accelerated MSSP algorithm on macroscopic road network modelling, and provides Contribution C6.

The third and final research question, regarding the potential shift in the choice of modelling

approach used by academics and modelling practitioners based on the relative improvements in simulation performance, was addressed by Chapter 6. It discussed the the impact of GPU acceleration from Chapters 3 to 5 on the modelling approach selection criteria. Additional benchmarking of the macroscopic road network simulation application was performed, using an adapted version of the procedurally generated grid road network, and the work of previous chapters, evidence is provided which suggests that microscopic models see a much greater impact from many-core acceleration. This is concluded to be due to the higher degree of parallelism exposed by the modelling approach, making it more suitable for highly-parallel processing hardware. This may shift the decision making process towards the finer-grained microscopic simulations, if GPUs are adopted by road network simulation packages, which are traditionally used due to the reduced computational cost.

7.1 Summary of Main Findings

In summary, this thesis set out to explore if the application of modern GPUs can be effectively be applied to both microscopic and macroscopic road network simulation approaches, and evaluate if the impact of GPUs on these modelling approaches may influence the often performance-related choice in modelling approach. Microscopic road network simulations were shown to be well suited for GPU parallelism, showing performance improvements of up to 67.7x for large scale simulations compared to a commercial multi-core CPU microsimulator. This was achieved through the definition of a benchmark model and scalable network, the development of a novel agent-communication pattern for many-core ABMs and the use of this communication pattern within a FLAME GPU based microscopic road network simulation, providing contributions C1, C2, C3 & C4. To explore the suitability of GPUs for macroscopic road network models, a novel many-source-shortest path algorithm for GPUs was developed and embedded within a commercial application, to demonstrate improved performance of up to 8.6x for macroscopic models, addressing contributions C5 and C6. Lastly, the relative improvements offered to each modelling approach were compared and evaluated, suggesting that GPUs may enable a shift towards the finer-grained microscopic modelling approach due to the increased degree of parallelism available. This provides contribution C7.

7.2 Future Work

This thesis has delivered work involving the benchmarking of microscopic and macroscopic road network simulations, plus a comparative performance assessment. However, there is room for further expansion on each aspect of this thesis.

The implemented microscopic road network simulations are intentionally simplified models,

with only a subset of features required to simulate real-world transport networks. A more complete ABM capable of road network simulations, leveraging the graph-based communication pattern could be explored, with scope for further many-core optimisation opportunities not exposed by the subset of models selected within this thesis. For instance, the communication pattern only considers a single edge of the graph at once, as this was sufficient for the benchmark model behaviours and scalable benchmark model used. Techniques already used within GPU ABM for spatial communication such as fixed-radius nearest-neighbour [263]–[265] searches could be adapted to provide messages from many elements of the graph, leveraging the memory hierarchy of the GPU to offer high performance.

With respect to the real-world macroscopic road network simulations of Chapter 5, the serial simulation phase of the application has become the performance limiting part of the application. Parallelising this, either through task-level or data-level parallelisation approaches, would bring further reduction to total application runtimes, but also provide scope for further advancements to the assignment phase of the application to be improved. Further improvements to the graph representation of the underlying transport network may also yield further performance improvements. Contraction hierarchies [199], [200] are already in use to improve the performance of shortest path calculations through a denser network representation, but there have been further developments to these techniques than present in SATURN, which may further improve the performance of the MSSP algorithm by increasing concurrency, and reducing the number of iterations required [201].

Both microscopic and macroscopic simulators do not benefit from GPU acceleration for small-scale simulations. However, as simulation ensembles are often required, or multiple scenarios are being evaluated, there may be opportunities to increase the degree of parallelism exposed to the GPU by using the many-cores to cooperatively work on multiple simulations concurrently. This would not only enable small-scale simulations to benefit from GPU acceleration, but also lead to reduced ensemble run-times, allowing broader evaluations to be conducted within the same time-frame.

Improved benchmark networks could also be investigated. Although road networks are very well-structured in some areas of the world, such as large American cities, other real-world networks are much more irregular in structure, having evolved organically as they grew over time. A more realistic benchmark network may offer a more accurate picture of the improvement of road network simulations on real-world use-cases than a regular grid-shaped benchmark. Scalable benchmark networks could also be designed with multiple modelling approaches in mind, to provide fairer performance evaluations.

One of the recent developments in GPUs is an increase in reduced-precision floating point arithmetic, primarily aimed at deep learning applications [83], [266], [267]. Since the Turing architecture, dedicated processing cores for performing mixed-precision matrix operations have

been introduced, and several formats of reduced precision floating point representations are now supported at the hardware level. One possible use-case for this is within the iterative assignment-simulation process of SATURN. As demonstrated in other domains such as molecular dynamics ([268], [269]), it's possible to use reduced precision formats during early iterations of convergent processes, switching to higher-precision formats as the approximation nears convergence. This could potentially be applied to the macroscopic assignment-simulation loop, to speed up the iterative process.

Processing hardware is also continuously evolving, with new GPU architecture such as Nvidia Ampere [266] offering higher levels of raw arithmetic performance, increased memory bandwidth and higher levels of parallelism all of which can result in improved performance to both simulation approaches. The balance of these performance characteristics may influence the algorithms and data structures which can be used within GPU accelerators. For instance, increasing the parallelism of the GPU will have implications on how to effectively use memory caches. Generational improvements to GPU memory will also lead to additional performance improvements, with kernels in both microscopic and macroscopic approaches being performance limited by memory bandwidth rather than compute power.

Acronyms

ABM

Agent Based Model 3, 4, 6, 8, 9, 28, 29, 33–35, 48–50, 58, 68, 74, 91, 140–143

ABS

Agent Based Simulation 35

AoS

Array of Structures 26

AoSoA

Array of Structures of Arrays 26

APSP

All Pairs Shortest Path 40, 45, 98, 100, 122

ASIC

Application-Specific Integrated Circuit 16

BGL

Boost Graph Library 45

CAV

Connected and Autonomous Vehicles 11, 28, 30

CPU

Central Processing Unit 2–5, 15–20, 26, 27, 32–35, 44–47, 49–51, 58, 86, 93, 95, 110, 113, 114, 116, 118, 125, 128, 132, 139

CSC

Compressed Sparse Column 14, 15

CSR

Compressed Sparse Row 14, 15, 62

CTA

Cooperative Thread Array 104, 141

CUDA

Compute Unified Device Architecture 58

DfT

Department for Transport 1

FLAME

Flexible Large-scale Agent Modelling Environment 34

FLAME GPU

Flexible Large-scale Agent Modelling Environment for Graphics Processing Unit 34, 35, 50, 58–60, 62, 65–68, 70, 71, 73–75, 140

FPGA

Field-Programmable Gate Array 16

GA

Genetic Algorithm 10

GPGPU

General Purpose Computing on Graphics Processing Units 2, 3, 6, 17–21, 112, 121

GPU

Graphics Processing Unit 2–4, 6, 7, 16–21, 23–27, 33–35, 47–50, 52, 58, 60, 65–68, 72, 73, 82, 83, 86, 87, 92, 94, 95, 101, 102, 107, 110, 112–116, 118, 125, 128, 135, 138–144

HGV

Heavy Goods Vehicle 11, 97, 108, 127

HPC

High Performance Computing 2, 15, 17, 18, 32

HTC

High Throughput Computing 15

ILP

Instruction Level Parallelism 16, 19

ITS

Intelligent Transport System 10

LGV

Light Goods Vehicle 11, 97, 127

MAS

Multi-Agent Systems 35

MIC

Many Integrated Core 18

MIMD

Multiple-Instruction-Multiple-Data 17

MISD

Multiple-Instruction-Single-Data 17

MPI

Message Passing Interface 20, 33, 34

MPMD

Multiple-Program-Multiple-Data 17

MSRP

Manufacturer's Suggested Retail Price 116

MSSP

Many Source Shortest Path 5, 94, 95, 100, 102, 103, 110–112, 115, 122, 130, 141, 143

OD

Origin Destination 29, 36, 38, 40

PBGL

Parallel Boost Graph Library 45

RTR

real-time-ratio 56, 84, 85, 91, 141

SATURN

Simulation and Assignment of Traffic to Urban Road Networks 46, 94, 95

SIMD

Single-Instruction-Multiple-Data 17, 33

SIMT

Single-Instruction-Multiple-Thread 17, 21, 24, 25, 34, 104, 106, 110, 122

SISD

Single-Instruction-Single-Data 17

SM

Streaming Multiprocessor 21–26, 77

SoA

Structure of Arrays 26, 77

SPMD

Single-Program-Multiple-Data 17

SRN

Strategic Road Network 11

SSSP

Single Source Shortest Path 40–42, 44–46, 94, 95, 97–100, 102, 110, 122, 141

TDP

Thermal Design Power 116

TfL

Transport for London 100

USD

United States Dollar 116

Bibliography

[1] Department for Transport, *Road traffic forecasts 2015*, `https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/260700/road-transport-forecasts-2013-extended-version.pdf`, Mar. 2015.

[2] UK Department for Transport, *Quarterly Road Traffic Estimates: Great Britain Quarter 4 (October - December) 2014*, Feb. 2015.

[3] H. Neffendorf, G. Fletcher, R. North, T. Worsley, and R. Bradley, *Modelling for intelligent mobility*, Feb. 2015.

[4] R. Shannon and J. D. Johannes, "Systems simulation: The art and science," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-6, no. 10, pp. 723–724, 1976. DOI: `10.1109/TSMC.1976.4309432`.

[5] E. Jonkers, J. Nellthorp, I. Wilmink, and J. Olstam, "Evaluation of eco-driving systems: A european analysis with scenarios and micro simulation," *Case Studies on Transport Policy*, 2018.

[6] R. Tu, I. Kamel, A. Wang, B. Abdulhai, and M. Hatzopoulou, "Development of a hybrid modelling approach for the generation of an urban on-road transportation emission inventory," *Transportation Research Part D: Transport and Environment*, vol. 62, pp. 604–618, 2018.

[7] M. Hadiuzzaman, T. Z. Qiu, and X.-Y. Lu, "Variable speed limit control design for relieving congestion caused by active bottlenecks," *Journal of Transportation Engineering*, vol. 139, no. 4, pp. 358–370, 2013.

[8] R. Liu, "The dracula dynamic network microsimulation model," in *Simulation approaches in transportation analysis*, Springer, 2005, pp. 23–56.

[9] E. Thonhofer, T. Palau, A. Kuhn, S. Jakubek, and M. Kozek, "Macroscopic traffic model for large scale urban traffic network design," *Simulation Modelling Practice and Theory*, vol. 80, pp. 32–49, 2018.

[10] H. Zhang and W. Jin, "Kinematic wave traffic flow model for mixed traffic," *Transportation Research Record*, vol. 1802, no. 1, pp. 197–204, 2002.

[11] W. Burghout, "Hybrid microscopic-mesoscopic traffic simulation," Ph.D. dissertation, KTH, 2004.

[12] C. Sommer, Z. Yao, R. German, and F. Dressler, "On the need for bidirectional coupling of road traffic microsimulation and network simulation," in *Proceedings of the 1st ACM SIGMOBILE Workshop on Mobility Models*, ser. MobilityModels '08, Hong Kong, Hong Kong, China: Association for Computing Machinery, 2008, pp. 41–48, ISBN: 9781605581118. DOI: 10.1145/1374688.1374697. [Online]. Available: https://doi.org/10.1145/1374688.1374697.

[13] G. Kotusevski and K. Hawick, "A review of traffic simulation software," in *Research Letters in the Information and Mathematical Sciences*, 13, Massey University, 2009, pp. 35–54.

[14] A. C, B. J, B. M, *et al.*, "Traffic simulation: Case for guidelines," Publications Office of the European Union, Luxembourg (Luxembourg), Scientific analysis or review, Technical guidance LB-NA-26534-EN-N, 2014. DOI: 10.2788/11382.

[15] G. Eliasson, "Modeling the experimentally organized economy: Complex dynamics in an empirical micro-macro model of endogenous economic growth," *Journal of Economic Behavior & Organization*, vol. 16, no. 1, pp. 153–182, 1991, ISSN: 0167-2681. DOI: https://doi.org/10.1016/0167-2681(91)90047-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0167268191900472.

[16] H. Rakha and M. Van Aerde, "Statistical analysis of day-to-day variations in real-time traffic flow data," *Transportation research record*, pp. 26–34, 1995.

[17] Top 500, *Top 500 website*, Last Accessed 2016-08-01, Aug. 2016. [Online]. Available: https://www.top500.org/.

[18] Green 500, *Green 500 website*, Last Accessed 2016-08-01, Aug. 2016. [Online]. Available: https://www.top500.org/green500/.

[19] S. Huang, S. Xiao, and W.-c. Feng, "On the energy efficiency of graphics processing units for scientific computing," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, IEEE, 2009, pp. 1–8.

[20] N. Collier and M. North, "Repast HPC: A platform for large-scale agent-based modeling," in *Large-Scale Computing*, John Wiley & Sons, Inc., Apr. 2012, pp. 81–109. DOI: 10.1002/9781118130506.ch5. [Online]. Available: https://doi.org/10.1002/9781118130506.ch5.

[21] G. Cordasco, C. Spagnuolo, and V. Scarano, "Toward the new version of d-mason: Efficiency, effectiveness and correctness in parallel and distributed agent-based simulations," in *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, IEEE, 2016, pp. 1803–1812.

[22] P. Richmond and M. K. Chimeh, "Flame gpu: Complex system simulation framework," in *2017 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, 2017, pp. 11–17.

[23] M. Zamith, M. Joselli, J. R. S. Junior, R. C. P. Leal-Toledo, E. Clua, and E. Soluri, "An approach for traffic forecast with gpu computing & cellular automata model," MediaLab, IC-UFF, Tech. Rep., 2012. [Online]. Available: `http://www.imago.ufpr.br/csbc2012/anais_csbc/eventos/gpu/artigos/GPU%20-%20An%20Approach%20for%20Traffic%20Forecast%20with%20GPU%20Computing%20Cellular%20Automata%20Model.pdf`.

[24] D. Strippgen and K. Nagel, "Multi-agent traffic simulation with cuda," in *2009 International Conference on High Performance Computing Simulation*, 2009, pp. 106–114. DOI: `10.1109/HPCSIM.2009.5192895`.

[25] J. Xiao, P. Andelfinger, D. Eckhoff, W. Cai, and A. Knoll, "Exploring execution schemes for agent-based traffic simulation on heterogeneous hardware," in *2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, IEEE, 2018, pp. 1–10.

[26] D. Rajf and T. Potuzak, "Comparison of road traffic simulation speed on cpu and gpu," in *2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, IEEE, 2019, pp. 1–8.

[27] P. Heywood, S. Maddock, R. Bradley, *et al.*, "A data-parallel many-source shortest-path algorithm to accelerate macroscopic transport network assignment," *Transportation Research Part C: Emerging Technologies*, vol. 104, pp. 332–347, 2019, ISSN: 0968-090X. DOI: `https://doi.org/10.1016/j.trc.2019.05.020`. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0968090X18303152`.

[28] P. Heywood, S. Maddock, J. Casas, D. Garcia, M. Brackstone, and P. Richmond, "Data-parallel agent-based microscopic road network simulation using graphics processing units," *Simulation Modelling Practice and Theory*, vol. 83, pp. 188–200, 2018, Agent-based Modelling and Simulation, ISSN: 1569-190X. DOI: `https://doi.org/10.1016/j.simpat.2017.11.002`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1569190X17301545`.

[29] P. Heywood, P. Richmond, and S. Maddock, "Road network simulation using flame gpu," in *Euro-Par 2015: Parallel Processing Workshops*, Springer, 2015, pp. 430–441. DOI: 10.1007/978-3-319-27308-2.

[30] D. Keenan, "M 62-m 1 interchange- a combined transyt/vissim micro-simulation assessment.," *Traffic engineering & control*, vol. 46, no. 5, pp. 178–187, 2005.

[31] D. Tsitsokas, M. Saeedmanesh, A. Kouvelas, and N. Geroliminis, "Optimal allocation of designated bus lanes in multi-modal urban networks," en, in *18th Swiss Transport Research Conference (STRC 2018)*, Conference Location: Ascona, Switzerland; Conference Date: May 16-18, 2018, Ascona: STRC, May 2018. DOI: 10.3929/ethz-b-000319611.

[32] A. Khattak, J. Yangsheng, and M. M. Abid, "Optimal configuration of the metro rail transit station service facilities by integrated simulation-optimization method using passengers' flow fluctuation," *Arabian Journal for Science and Engineering*, pp. 1–18, 2018.

[33] S. McMillan, A. Nicholson, and G. Koorey, "Incident management modelling using microsimulation with adaptive signal control," *10th ITS Asia-Pacific Forum*, Jul. 2009. [Online]. Available: http://hdl.handle.net/10092/3117.

[34] L. Cruz-Piris, D. Rivera, S. Fernandez, and I. Marsa-Maestre, "Optimized sensor network and multi-agent decision support for smart traffic light management," *Sensors*, vol. 18, no. 2, p. 435, 2018.

[35] J. S. Kaizer, A. K. Heller, and W. L. Oberkampf, "Scientific computer simulation review," *Reliability Engineering & System Safety*, vol. 138, pp. 210–218, 2015.

[36] O. F. Lange, D. Van der Spoel, and B. L. De Groot, "Scrutinizing molecular mechanics force fields on the submicrosecond timescale with nmr data," *Biophysical journal*, vol. 99, no. 2, pp. 647–655, 2010.

[37] W. Kinzel, "Phase transitions of cellular automata," *Zeitschrift für Physik B Condensed Matter*, vol. 58, no. 3, pp. 229–244, 1985.

[38] A. K. Seth, "Measuring autonomy and emergence via granger causality," *Artificial life*, vol. 16, no. 2, pp. 179–196, 2010.

[39] R. M. Fujimoto, "Parallel discrete event simulation," *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, 1990.

[40] N. Matloff, "Introduction to discrete-event simulation and the simpy language," *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, vol. 2, no. 2009, pp. 1–33, 2008.

[41] W. G. Wilson, "Resolving discrepancies between deterministic population models and individual-based simulations," *The American Naturalist*, vol. 151, no. 2, pp. 116–134, 1998.

[42] M. J. Lighthill and G. B. Whitham, "On kinematic waves ii. a theory of traffic flow on long crowded roads," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 229, no. 1178, pp. 317–345, 1955.

[43] B. Haut, G. Bastin, and Y. Chitour, "A macroscopic traffic model for road networks with a representation of the capacity drop phenomenon at the junctions," *IFAC Proceedings Volumes*, vol. 38, no. 1, pp. 114–119, 2005.

[44] M. Papageorgiou, "Some remarks on macroscopic traffic flow modelling," *Transportation Research Part A: Policy and Practice*, vol. 32, no. 5, pp. 323–329, 1998, ISSN: 0965-8564. DOI: https://doi.org/10.1016/S0965-8564(97)00048-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965856497000487.

[45] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems," *Proceedings of the National Academy of Sciences*, vol. 99, no. Supplement 3, pp. 7280–7287, May 2002. DOI: 10.1073/pnas.082080899. [Online]. Available: https://doi.org/10.1073/pnas.082080899.

[46] K.-O. Kim and L. R. Rilett, "Simplex-based calibration of traffic microsimulation models with intelligent transportation systems data," *Transportation Research Record*, vol. 1855, no. 1, pp. 80–89, 2003. DOI: 10.3141/1855-10. eprint: https://doi.org/10.3141/1855-10. [Online]. Available: https://doi.org/10.3141/1855-10.

[47] R. Borsche and A. Meurer, "Microscopic and macroscopic models for coupled car traffic and pedestrian flow," *Journal of Computational and Applied Mathematics*, vol. 348, pp. 356–382, 2019.

[48] J. Casas, J. Perarnau, and A. Torday, "The need to combine different traffic modelling levels for effectively tackling large-scale projects adding a hybrid meso/micro approach," *Procedia-Social and Behavioral Sciences*, vol. 20, pp. 251–262, 2011.

[49] L. Montero, E. Codina, J. Barceló, and P. Barceló, "Combining macroscopic and microscopic approaches for transportation planning and design of road networks," in *Proceedings of the 19th ARRB Meeting, Sydney*, Citeseer, 1998.

[50] T. Trucano, L. Swiler, T. Igusa, W. Oberkampf, and M. Pilch, "Calibration, validation, and sensitivity analysis: What's what," *Reliability Engineering & System Safety*, vol. 91, no. 10, pp. 1331–1357, 2006, The Fourth International Conference on Sensitivity Analysis of Model Output (SAMO 2004), ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2005.11.031. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832005002437.

[51] S. Crouch, N. P. C. Hong, S. Hettrick, *et al.*, “The software sustainability institute: Changing research software attitudes and practices.,” *Comput. Sci. Eng.*, vol. 15, no. 6, pp. 74–80, 2013.

[52] G. Wilson, D. A. Aruliah, C. T. Brown, *et al.*, “Best practices for scientific computing,” *PLoS biology*, vol. 12, no. 1, e1001745, 2014.

[53] R. C. Jiménez, M. Kuzak, M. Alhamdoosh, *et al.*, “Four simple recommendations to encourage best practices in research software,” *F1000Research*, vol. 6, 2017.

[54] K. M. Carley, “Validating computational models,” Carnegie Mellon University, School of Computer Science, Institute for Software Research, Tech. Rep., 2017, CMU-ISR-17-105.

[55] S.-J. Kim, W. Kim, and L. R. Rilett, “Calibration of microsimulation models using nonparametric statistical techniques,” *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1935, no. 1, pp. 111–119, Jan. 2005. DOI: `10.1177/0361198105193500113`. [Online]. Available: `https://doi.org/10.1177/0361198105193500113`.

[56] Y. Hollander and R. Liu, “The principles of calibrating traffic microsimulation models,” *Transportation*, vol. 35, no. 3, pp. 347–362, Jan. 2008. DOI: `10.1007/s11116-007-9156-2`. [Online]. Available: `https://doi.org/10.1007/s11116-007-9156-2`.

[57] J. A. Nelder and R. Mead, “A simplex method for function minimization,” *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[58] B. Calvez and G. Hutzler, “Parameter space exploration of agent-based models,” in *international conference on knowledge-based and intelligent information and engineering systems*, Springer, 2005, pp. 633–639.

[59] L. Chu, H. X. Liu, J.-S. Oh, and W. Recker, “A calibration procedure for microscopic traffic simulation,” in *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, IEEE, vol. 2, 2003, pp. 1574–1579.

[60] D. Whitley, “A genetic algorithm tutorial,” *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[61] S. Forrest, “Genetic algorithms,” *ACM Computing Surveys*, vol. 28, no. 1, pp. 77–80, Mar. 1996. DOI: `10.1145/234313.234350`. [Online]. Available: `https://doi.org/10.1145/234313.234350`.

[62] P. Knepell and D. Arangno, *Simulation Validation: A Confidence Assessment Methodology*, ser. IEEE Computer Society Press monograph. Wiley, 1993, ISBN: 9780818635120.

[63] M. M. Ishaque and R. B. Noland, “Pedestrian and vehicle flow calibration in multimodal traffic microsimulation,” *Journal of Transportation Engineering*, vol. 135, no. 6, pp. 338–348, Jun. 2009. DOI: `10.1061/(asce)0733-947x(2009)135:6(338)`. [Online]. Available: `https://doi.org/10.1061/(asce)0733-947x(2009)135:6(338)`.

[64] Department for Transport, *Strategic Road Network Statistics*, 2015. [Online]. Available: https://www.gov.uk/government/statistics/strategic-road-network-statistics.

[65] Department for Transport, *GB Road Traffic Counts*, Last Accessed 2015-02-25. [Online]. Available: http://data.gov.uk/dataset/gb-road-traffic-counts.

[66] A. V. Goldberg, "Finding a maximum density subgraph," EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-84-171, 1984. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1984/5956.html.

[67] J. Hartmanis and R. E. Stearns, "On the computational complexity of algorithms," *Transactions of the American Mathematical Society*, vol. 117, pp. 285–306, 1965.

[68] R. Schaller, "Moore's law: Past, present and future," *IEEE Spectrum*, vol. 34, no. 6, pp. 52–59, 1997. DOI: 10.1109/6.591665.

[69] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974. DOI: 10.1109/JSSC.1974.1050511.

[70] K. Asanović, R. Bodik, B. C. Catanzaro, *et al.*, "The landscape of parallel computing research: A view from berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec. 2006. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html.

[71] D. P. Rodgers, "Improvements in multiprocessor system design," *ACM SIGARCH Computer Architecture News*, vol. 13, no. 3, pp. 225–231, Jun. 1985. DOI: 10.1145/327070.327215. [Online]. Available: https://doi.org/10.1145/327070.327215.

[72] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008. DOI: 10.1109/MC.2008.209.

[73] J. L. Gustafson, "Reevaluating amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[74] S. Chatterjee, S. Tasırlar, Z. Budimlic, *et al.*, "Integrating asynchronous task parallelism with mpi," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IEEE, 2013, pp. 712–725.

[75] W. D. Hillis and G. L. Steele, "Data parallel algorithms," *Commun. ACM*, vol. 29, no. 12, pp. 1170–1183, Dec. 1986, ISSN: 0001-0782. DOI: 10.1145/7902.7903. [Online]. Available: https://doi.org/10.1145/7902.7903.

[76] D. W. Wall, "Limits of instruction-level parallelism," in *Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, 1991, pp. 176–188.

[77] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, "Single-chip heterogeneous computing: Does the future include custom logic, fpgas, and gpgpus?" In *2010 43rd annual IEEE/ACM international symposium on microarchitecture*, IEEE, 2010, pp. 225–236.

[78] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE transactions on computers*, vol. 100, no. 9, pp. 948–960, 1972.

[79] NVIDIA, *Nvidia tesla v100 gpu architecture*, Online, 2017. [Online]. Available: `https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf`.

[80] F. Darema, D. A. George, V. A. Norton, and G. F. Pfister, "A single-program-multiple-data computational model for epex/fortran," *Parallel Computing*, vol. 7, no. 1, pp. 11–24, 1988.

[81] Top 500, *Top500 list - november 2020*, Last Accessed 2021-02-02, Nov. 2020. [Online]. Available: `https://www.top500.org/lists/top500/list/2020/11/`.

[82] D. Mayhew and V. Krishnan, "Pci express and advanced switching: Evolutionary path to building next generation interconnects," in *11th Symposium on High Performance Interconnects, 2003. Proceedings.*, IEEE, 2003, pp. 21–29.

[83] NVIDIA, *NVIDIA Tesla P100 (whitepaper)*, Online, 2016. [Online]. Available: `https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf`.

[84] A. Duran and M. Klemm, "The intel® many integrated core architecture," in *2012 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, 2012, pp. 365–366.

[85] J. Jeffers and J. Reinders, *Intel Xeon Phi Coprocessor High Performance Programming*, 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013, ISBN: 9780124104143.

[86] G. Teodoro, T. Kurc, J. Kong, L. Cooper, and J. Saltz, "Comparative performance analysis of intel (r) xeon phi (tm), gpu, and cpu: A case study from microscopy image analysis," in *2014 IEEE 28th international parallel and distributed processing symposium*, IEEE, 2014, pp. 1063–1072.

[87] A. Williams, *C++ concurrency in action*. Shelter Island, NY: Manning Publications, 2019, ISBN: 9781617294693.

[88] B. Barney, *Posix threads programming*, Lawrence Livermore National Laboratory, UCRL-MI-133316, 2009. [Online]. Available: `https://hpc-tutorials.llnl.gov/posix/`.

[89] A. Edelman, "Julia: A fresh approach to parallel programming," in *2015 IEEE International Parallel and Distributed Processing Symposium*, IEEE, 2015, pp. 517–517.

[90] B. O'Neal, *Using c++17 parallel algorithms for better performance*, Last Accessed 2021-01-26, Oct. 31, 2018. [Online]. Available: https://devblogs.microsoft.com/cppblog/using-c17-parallel-algorithms-for-better-performance/.

[91] NVIDIA, *Nvidia hpc sdk version 20.11 documentation*, 2020. [Online]. Available: https://docs.nvidia.com/hpc-sdk/archive/20.11/ (visited on 02/01/2021).

[92] D. Merrill, "Cub: A pattern of "collective" software design, abstraction, and reuse for kernel-level programming," S5617 GPU Technology Conference 2015, 2015. [Online]. Available: https://on-demand.gputechconf.com/gtc/2015/presentation/S5617-Duane-Merrill.pdf.

[93] N. Bell and J. Hoberock, "Thrust: A productivity-oriented library for cuda," in *GPU computing gems Jade edition*, Elsevier, 2012, pp. 359–371.

[94] J. Szuppe, "Boost. compute: A parallel computing library for c++ based on opencl," in *Proceedings of the 4th International Workshop on OpenCL*, 2016, pp. 1–39.

[95] L. Dagum and R. Menon, "Openmp: An industry standard api for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[96] A. Mishra, L. Li, M. Kong, H. Finkel, and B. Chapman, "Benchmarking and evaluating unified memory for openmp gpu offloading," in *Proceedings of the Fourth Workshop on the LLVM Compiler Infrastructure in HPC*, 2017, pp. 1–10.

[97] OpenACC-Standard.org, *The openacc application programming interface, v1.0*, Nov. 2011. [Online]. Available: https://www.openacc.org/sites/default/files/inline-files/OpenACC_1_0_specification.pdf.

[98] T. Ni, "Direct compute: Bring gpu computing to the mainstream," GPU Technology Conference 2009, 2009. [Online]. Available: https://www.nvidia.com/content/GTC/documents/1015_GTC09.pdf.

[99] D. Shreiner, G. Sellers, J. Kessenich, and B. Licea-Kane, *OpenGL programming guide: The Official guide to learning OpenGL, version 4.3.* Addison-Wesley, 2013.

[100] G. Sellers and J. Kessenich, *Vulkan programming guide: The official guide to learning vulkan.* Addison-Wesley Professional, 2016.

[101] Khronos Group, *About OpenCL - Khronos Group*, Last Accessed 2015-03-25. [Online]. Available: https://www.khronos.org/opencl/.

[102] T. Deakin and S. McIntosh-Smith, "Evaluating the performance of hpc-style sycl applications," in *Proceedings of the International Workshop on OpenCL*, 2020, pp. 1–11.

[103] NVIDIA Corporation, *CUDA Parallel Programming and Computing Platform*, Last Accessed 2015-03-25. [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html.

[104] S. Memeti, L. Li, S. Pllana, J. Kołodziej, and C. Kessler, "Benchmarking opencl, openacc, openmp, and cuda: Programming productivity, performance, and energy consumption," in *Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*, 2017, pp. 1–6.

[105] T. Hoshino, N. Maruyama, S. Matsuoka, and R. Takaki, "Cuda vs openacc: Performance case studies with kernel benchmarks and a memory-bound cfd application," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, IEEE, 2013, pp. 136–143.

[106] R. Reyes, I. López, J. J. Fumero, and F. de Sande, "A preliminary evaluation of openacc implementations," *The Journal of Supercomputing*, vol. 65, no. 3, pp. 1063–1075, 2013.

[107] Message Passing Interface Forum, *Mpi: A message-passing interface standard version 3.1*, USA: University of Tennessee, 2015. [Online]. Available: https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[108] E. Gabriel, G. E. Fagg, G. Bosilca, *et al.*, "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, Springer, 2004, pp. 97–104.

[109] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the mpi message passing interface standard," *Parallel computing*, vol. 22, no. 6, pp. 789–828, 1996.

[110] D. K. Panda, H. Subramoni, C.-H. Chu, and M. Bayatpour, "The mvapich project: Transforming research into high-performance mpi library for hpc community," *Journal of Computational Science*, p. 101 208, 2020.

[111] S. Potluri, K. Hamidouche, A. Venkatesh, D. Bureddy, and D. K. Panda, "Efficient internode mpi communication using gpudirect rdma for infiniband clusters with nvidia gpus," in *2013 42nd International Conference on Parallel Processing*, IEEE, 2013, pp. 80–89.

[112] C. Nvidia, *Cuda c++ programming guide*, Last Accessed 2021-02-02, Jan. 2021. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

[113] N. Weber and M. Goesele, "Matog: Array layout auto-tuning for cuda," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 3, pp. 1–26, 2017.

[114] K. Bernhardt, "Agent-based modeling in transportation," *Artificial Intelligence in Transportation*, pp. 72–80, Jan. 2007.

[115] C. Z. Renner, T. W. Nicolai, and K. Nagel, "Agent-based land use transport interaction modeling: State of the art," *Institute of Technology (TU Berlin)*, 2014.

[116] J. J. Sánchez-Medina, M. J. Galán-Moreno, and E. Rubio-Royo, "Traffic signal optimization in "la almozara" district in saragossa under congestion conditions, using genetic algorithms, traffic microsimulation, and cluster computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 1, pp. 132–141, 2009.

[117] H. Lu, G. Song, and L. Yu, "The "acceleration cliff": An investigation of the possible error source of the vsp distributions generated by wiedemann car-following model," *Transportation Research Part D: Transport and Environment*, vol. 65, pp. 161–177, 2018.

[118] Z. Wen-Xing and Z. Li-Dong, "A new car-following model for autonomous vehicles flow with mean expected velocity field," *Physica A: Statistical Mechanics and its Applications*, vol. 492, pp. 2154–2165, 2018.

[119] H. Ou and T.-Q. Tang, "An extended two-lane car-following model accounting for inter-vehicle communication," *Physica A: Statistical Mechanics and its Applications*, vol. 495, pp. 260–268, 2018.

[120] P. A. Lopez, M. Behrisch, L. Bieker-Walz, *et al.*, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*, IEEE, 2018. [Online]. Available: `https://elib.dlr.de/124092/`.

[121] M. Fellendorf, "Vissim: A microscopic simulation tool to evaluate actuated signal control including bus priority," in *64th Institute of Transportation Engineers Annual Meeting*, Springer, 1994, pp. 1–9.

[122] J. Barceló and J. Casas, "Dynamic network simulation with aimsun," in *Simulation approaches in transportation analysis*, Springer, 2005, pp. 57–98.

[123] J. Jin, X. Ma, and I. Kosonen, "An intelligent control system for traffic lights with simulation-based evaluation," *Control engineering practice*, vol. 58, pp. 24–33, 2017.

[124] K. Gajananan, S. Sontisirikit, J. Zhang, *et al.*, "A cooperative its study on green light optimisation using an integrated traffic, driving, and communication simulator," in *Australasian Transport Research Forum 2013 Proceedings*, Australasian Transport Research Forum (ATRF), 2013.

[125] A. Sciarretta, G. De Nunzio, and L. L. Ojeda, "Optimal ecodriving control: Energy-efficient driving of road vehicles as an optimal control problem," *IEEE Control Systems Magazine*, vol. 35, no. 5, pp. 71–90, 2015.

[126] P. Gipps, “A model for the structure of lane-changing decisions,” *Transportation Research Part B: Methodological*, vol. 20, no. 5, pp. 403–414, 1986, ISSN: 0191-2615. DOI: `https://doi.org/10.1016/0191-2615(86)90012-3`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/0191261586900123`.

[127] G. Lee, “A generalization of linear car-following theory,” *Operations Research*, vol. 14, no. 4, pp. 595–606, Aug. 1966. DOI: `10.1287/opre.14.4.595`. [Online]. Available: `https://doi.org/10.1287/opre.14.4.595`.

[128] P. Gipps, “A behavioural car-following model for computer simulation,” *Transportation Research Part B: Methodological*, vol. 15, no. 2, pp. 105–111, 1981, ISSN: 0191-2615. DOI: `https://doi.org/10.1016/0191-2615(81)90037-0`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/0191261581900370`.

[129] M. Treiber, A. Hennecke, and D. Helbing, “Congested traffic states in empirical observations and microscopic simulations,” *Physical Review E*, vol. 62, no. 2, pp. 1805–1824, Aug. 2000, ISSN: 1095-3787. DOI: `10.1103/physreve.62.1805`. [Online]. Available: `http://dx.doi.org/10.1103/PhysRevE.62.1805`.

[130] S. Krauss, P. Wagner, and C. Gawron, “Metastable states in a microscopic model of traffic flow,” *Phys. Rev. E*, vol. 55, pp. 5597–5602, 5 May 1997. DOI: `10.1103/PhysRevE.55.5597`. [Online]. Available: `https://link.aps.org/doi/10.1103/PhysRevE.55.5597`.

[131] B. Higgs, M. Abbas, and A. Medina, “Analysis of the wiedemann car following model over different speeds using naturalistic data,” in *Procedia of RSS Conference*, 2011, pp. 1–22.

[132] R. Wiedemann, “Simulation des straßenverkehrsflusses,” *Schriftenreihe des IfV, Institut für Verkehrswesen. Universität Karlsruhe*, 1974, In German.

[133] B. S. Kerner and H. Rehborn, “Experimental properties of phase transitions in traffic flow,” *Physical Review Letters*, vol. 79, no. 20, p. 4030, 1997.

[134] V. Milanés and S. E. Shladover, “Modeling cooperative and autonomous adaptive cruise control dynamic responses using experimental data,” *Transportation Research Part C: Emerging Technologies*, vol. 48, pp. 285–300, 2014.

[135] L. Xiao, M. Wang, and B. van Arem, “Realistic car-following models for microscopic simulation of adaptive and cooperative adaptive cruise control vehicles,” *Transportation Research Record*, vol. 2623, no. 1, pp. 1–9, 2017.

[136] J. Zhang, T.-Q. Tang, and S.-W. Yu, “An improved car-following model accounting for the preceding car’s taillight,” *Physica A: Statistical Mechanics and its Applications*, vol. 492, pp. 1831–1837, 2018.

[137] X. Huang, J. Sun, and J. Sun, "A car-following model considering asymmetric driving behavior based on long short-term memory neural networks," *Transportation Research Part C: Emerging Technologies*, vol. 95, pp. 346–362, 2018.

[138] H. Farah, S. Bekhor, A. Polus, and T. Toledo, "A passing gap acceptance model for two-lane rural highways," *Transportmetrica*, vol. 5, no. 3, pp. 159–172, 2009.

[139] A. Kesting, M. Treiber, and D. Helbing, "General lane-changing model MOBIL for car-following models," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1999, no. 1, pp. 86–94, Jan. 2007. DOI: `10.3141/1999-10`. [Online]. Available: `https://doi.org/10.3141/1999-10`.

[140] J. A. Laval and L. Leclercq, "Microscopic modeling of the relaxation phenomenon using a macroscopic lane-changing model," *Transportation Research Part B: Methodological*, vol. 42, no. 6, pp. 511–522, 2008.

[141] T. Toledo, C. F. Choudhury, and M. E. Ben-Akiva, "Lane-changing model with explicit target lane choice," *Transportation Research Record*, vol. 1934, no. 1, pp. 157–165, 2005.

[142] D.-F. Xie, Z.-Z. Fang, B. Jia, and Z. He, "A data-driven lane-changing model based on deep learning," *Transportation research part C: emerging technologies*, vol. 106, pp. 41–60, 2019.

[143] J. Barceló, E. Codina, J. Casas, J. L. Ferrer, and D. García, "Microscopic traffic simulation: A tool for the design, analysis and evaluation of intelligent transport systems," *Journal of Intelligent and Robotic Systems*, vol. 41, no. 2-3, pp. 173–203, Jan. 2005. DOI: `10.1007/s10846-005-3808-2`. [Online]. Available: `https://doi.org/10.1007/s10846-005-3808-2`.

[144] Transport Simulation Systems, *Aimsun 8 users' manual*, Online version provided by a third party, 2014. [Online]. Available: `https://eclass.upatras.gr/modules/document/file.php/CIV1716/Aimsun%20Users%20Manual%20v7.pdf`.

[145] M. Hamed, S. Easa, and R. Batayneh, "Disaggregate gap-acceptance model for unsignalized t-intersections," *Journal of transportation engineering*, vol. 123, no. 1, pp. 36–42, 1997.

[146] R. J. Troutbeck and S. Kako, "Limited priority merge at unsignalized intersections," *Transportation Research Part A: Policy and Practice*, vol. 33, no. 3-4, pp. 291–304, 1999.

[147] E. Aakre and A. Aakre, "Modeling cooperation in unsignalized intersections," *Procedia Computer Science*, vol. 109, pp. 875–880, 2017.

[148] S. Y. Hwang and C. H. Park, "Modeling of the gap acceptance behavior at a merging section of urban freeway," in *Proceedings of the Eastern Asia Society for Transportation Studies*, Citeseer, vol. 5, 2005, e1656.

[149] G. Lee, "Modeling gap acceptance at freeway merges," Ph.D. dissertation, Massachusetts Institute of Technology, 2006.

[150] R. Vaiana, V. Gallelli, and T. Iuele, "Methodological approach for evaluation of roundabout performances through microsimulation," in *Applied Mechanics and Materials*, Trans Tech Publ, vol. 253, 2013, pp. 1956–1966.

[151] M. Dutta and M. A. Ahmed, "Critical gaps at three-legged unsignalised intersections using microsimulation," in *Proceedings of the Institution of Civil Engineers-Transport*, Thomas Telford Ltd, 2020, pp. 1–10.

[152] D. I. Robertson and R. D. Bretherton, "Optimizing networks of traffic signals in real time-the scoot method," *IEEE Transactions on vehicular technology*, vol. 40, no. 1, pp. 11–15, 1991.

[153] E. Richardson, P. Davies, and D. Newell, "Modelling a smart motorway," in *Conference on Complex, Intelligent, and Software Intensive Systems*, Springer, 2019, pp. 148–158.

[154] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal on Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, Dec. 2012. [Online]. Available: `https://sumo.dlr.de/pdf/sysmea_v5_n34_2012_4.pdf`.

[155] M. Balmer, M. Rieser, K. Meister, D. Charypar, N. Lefebvre, and K. Nagel, "Matsim-t: Architecture and simulation times," in *Multi-agent systems for traffic and transportation engineering*, IGI Global, 2009, pp. 57–78.

[156] G. D. Cameron and G. I. Duncan, "Paramics—parallel microscopic simulation of road traffic," *The Journal of Supercomputing*, vol. 10, no. 1, pp. 25–53, 1996.

[157] P. Yedavalli, K. Kumar, and P. Waddell, "Microsimulation analysis for network traffic assignment (manta) at metropolitan-scale for agile transportation planning," *arXiv preprint arXiv:2007.03614*, 2020.

[158] K. Wang and Z. Shen, "A gpu-based parallel genetic algorithm for generating daily activity plans," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1474–1480, 2012. DOI: `10.1109/TITS.2012.2205147`.

[159] H. Zhou, J. Dorsman, M. Snelder, E. de Romph, and M. Mandjes, "Gpu-based parallel computing for activity-based travel demand models," *Procedia Computer Science*, vol. 151, pp. 726–732, 2019.

[160] K. Wang and Z. Shen, "Artificial societies and gpu-based cloud computing for intelligent transportation management," *IEEE Intelligent Systems*, vol. 26, no. 4, pp. 22–28, 2011. DOI: `10.1109/MIS.2011.65`.

[161] Y. Sano and N. Fukuta, "A gpu-based framework for large-scale multi-agent traffic simulations," in *Advanced Applied Informatics (IIAIAAI), 2013 IIAI International Conference on*, IEEE, 2013, pp. 262–267.

[162] Y. Xu, G. Tan, X. Li, and X. Song, "Mesoscopic traffic simulation on cpu/gpu," in *Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ACM, 2014, pp. 39–50.

[163] X. Song, Z. Xie, Y. Xu, *et al.*, "Supporting real-world network-oriented mesoscopic traffic simulation on gpu," *Simulation Modelling Practice and Theory*, vol. 74, pp. 46–63, 2017.

[164] A. Saprykin, N. Chokani, and R. S. Abhari, "Gemsim: A gpu-accelerated multi-modal mobility simulator for large-scale scenarios," *Simulation Modelling Practice and Theory*, vol. 94, pp. 199–214, 2019.

[165] V. A. Vu and G. Tan, "High-performance mesoscopic traffic simulation with gpu for large scale networks," in *2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, IEEE, 2017, pp. 1–9.

[166] G. Cordasco, R. D. Chiara, A. Mancuso, D. Mazzeo, V. Scarano, and C. Spagnuolo, "A framework for distributing agent-based simulations," in *Euro-Par Workshops (1)'11*, 2011, pp. 460–470.

[167] S. Tisue and U. Wilensky, "Netlogo: A simple environment for modeling complexity," in *International conference on complex systems*, Boston, MA, vol. 21, 2004, pp. 16–21.

[168] M. J. North, N. T. Collier, and J. R. Vos, "Experiences creating three implementations of the repast agent modeling toolkit," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 16, no. 1, pp. 1–25, 2006.

[169] L. Chin, D. Worth, C. Greenough, S. Coakley, M. Holcombe, and M. Gheorghe, "Flame-ii : A redesign of the flexible large-scale agent-based modelling environment," *Rutherford Appleton Laboratory Technical Reports*, 2012.

[170] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, and C. Greenough, "Exploitation of high performance computing in the flame agent-based simulation framework," in *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, IEEE, 2012, pp. 538–545.

[171] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "Mason: A multiagent simulation environment," *Simulation*, vol. 81, no. 7, pp. 517–527, 2005.

[172] G. Cordasco, A. Mancuso, F. Milone, and C. Spagnuolo, "Communication strategies in distributed agent-based simulations: The experience with d-mason," in *Euro-Par Workshops'13*, 2013, pp. 533–543.

[173] G. Cordasco, V. Scarano, and C. Spagnuolo, "Distributed mason: A scalable distributed multi-agent simulation environment," *Simulation Modelling Practice and Theory*, 2018.

[174] M. J. North, N. T. Collier, J. Ozik, *et al.*, "Complex adaptive systems modeling with repast simphony," *Complex Adaptive Systems Modeling*, vol. 1, no. 1, Mar. 2013. DOI: 10.1186/2194-3206-1-3. [Online]. Available: https://doi.org/10.1186/2194-3206-1-3.

[175] N. Collier and M. North, "Parallel agent-based simulation with repast for high performance computing," *SIMULATION*, vol. 89, no. 10, pp. 1215–1235, Nov. 2012. DOI: 10.1177/0037549712462620. [Online]. Available: https://doi.org/10.1177/0037549712462620.

[176] S. Coakley, P. Richmond, M. Gheorghe, *et al.*, "Large-scale simulations with flame," in *Intelligent Agents in Data-intensive Computing*, Springer, 2016, pp. 123–142.

[177] M. Holcombe, "X-machines as a basis for dynamic system specification," *Software Engineering Journal*, vol. 3, no. 2, pp. 69–76, 1988.

[178] P. Richmond, D. Walker, S. Coakley, and D. Romano, "High performance cellular level agent-based simulation with FLAME for the GPU," *Briefings in Bioinformatics*, vol. 11, no. 3, pp. 334–347, Feb. 2010. DOI: 10.1093/bib/bbp073. [Online]. Available: https://doi.org/10.1093/bib/bbp073.

[179] P. Richmond, "Flame gpu technical report and user guide (cs-11-03)," Technical report, Department of Computer Science, University of Sheffield, Tech. Rep., 2011.

[180] G. Laville, K. Mazouzi, C. Lang, N. Marilleau, B. Herrmann, and L. Philippe, "Mcmas: A toolkit to benefit from many-core architecure in agent-based simulation," in *European conference on parallel processing*, Springer, 2013, pp. 544–554.

[181] P. Richmond and D. Romano, "Template-driven agent-based modeling and simulation with CUDA," in *GPU Computing Gems Emerald Edition*, Elsevier, 2011, pp. 313–324. DOI: 10.1016/b978-0-12-384988-5.00021-8. [Online]. Available: https://doi.org/10.1016/b978-0-12-384988-5.00021-8.

[182] T. Karmakham, P. Richmond, and D. M. Romano, *Agent-based large scale simulation of pedestrians with adaptive realistic navigation vector fields*, en, 2010. DOI: 10.2312/LOCALCHAPTEREVENTS/TPCG/TPCG10/067-074. [Online]. Available: http://diglib.eg.org/handle/10.2312/LocalChapterEvents.TPCG.TPCG10.067-074.

[183] S. Tamrakar, P. Richmond, and R. M. D'Souza, "PI-FLAME: A parallel immune system simulator using the FLAME graphic processing unit environment," *SIMULATION*, vol. 93, no. 1, pp. 69–84, Oct. 2016. DOI: 10.1177/0037549716673724. [Online]. Available: https://doi.org/10.1177/0037549716673724.

[184] M. K. Chimeh and P. Richmond, “Simulating heterogeneous behaviours in complex systems on gpus,” *Simulation Modelling Practice and Theory*, vol. 83, pp. 3–17, 2018.

[185] J. Pyle, M. K. Chimeh, and P. Richmond, “Surrogate modelling for efficient discovery of emergent population dynamics,” in *2019 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, 2019, pp. 99–106.

[186] B. Cosenza, N. Popov, B. Juurlink, *et al.*, “OpenABL: A domain-specific language for parallel and distributed agent-based simulations,” in *European Conference on Parallel Processing*, Springer, 2018, pp. 505–518.

[187] D. Milne and D. V. Vliet, “Implementing road user charging in saturn,” ARRAY(0x55e735d1c900), Leeds, UK, Working Paper Working Paper 410, Dec. 1993, Copyright of the Institute of Transport Studies, University Of Leeds. [Online]. Available: `https://eprints.whiterose.ac.uk/2167/`.

[188] L. J. LeBlanc, E. K. Morlok, and W. P. Pierskalla, “An efficient approach to solving the road network equilibrium traffic assignment problem,” *Transportation research*, vol. 9, no. 5, pp. 309–318, 1975.

[189] M. Hall, D. Vliet, and L. Willumsen, “Saturn—a simulation-assignment model for the evaluation of traffic management schemes,” *Traffic Engineering and Control*, vol. 21, pp. 168–176, Apr. 1980.

[190] J. G. Wardrop, “Road paper. some theoretical aspects of road traffic research,” *Proceedings of the Institution of Civil Engineers*, vol. 1, no. 3, pp. 325–362, May 1952. DOI: `10.1680/ipeds.1952.11259`. [Online]. Available: `https://doi.org/10.1680/ipeds.1952.11259`.

[191] M. Frank, P. Wolfe, *et al.*, “An algorithm for quadratic programming,” *Naval research logistics quarterly*, vol. 3, no. 1-2, pp. 95–110, 1956.

[192] M. Fukushima, “A modified frank-wolfe algorithm for solving the traffic assignment problem,” *Transportation Research Part B: Methodological*, vol. 18, no. 2, pp. 169–177, 1984.

[193] G. Gentile and K. Noekel, “Linear user cost equilibrium: The new algorithm for traffic assignment in visum,” in *Proceedings of European Transport Conference 2009*, 2009.

[194] M. Mitradjieva and P. O. Lindberg, “The stiff is moving—conjugate direction frank-wolfe methods with applications to traffic assignment,” *Transportation Science*, vol. 47, no. 2, pp. 280–293, 2013.

[195] J. D. Abernethy and J.-K. Wang, "On frank-wolfe and equilibrium computation," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/7371364b3d72ac9a3ed8638e6f0be2c9-Paper.pdf.

[196] H. Bar-Gera, "Origin-based algorithm for the traffic assignment problem," *Transportation Science*, vol. 36, no. 4, pp. 398–417, 2002, ISSN: 00411655, 15265447. [Online]. Available: http://www.jstor.org/stable/25769124.

[197] R. B. Dial, "A path-based user-equilibrium traffic assignment algorithm that obviates path storage and enumeration," *Transportation Research Part B: Methodological*, vol. 40, no. 10, pp. 917–936, 2006.

[198] D. Van Vliet, "Improved shortest path algorithms for transport networks," *Transportation Research*, vol. 12, no. 1, pp. 7–20, 1978.

[199] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *International Workshop on Experimental and Efficient Algorithms*, Springer, 2008, pp. 319–333.

[200] J. Dibbelt, B. Strasser, and D. Wagner, "Customizable contraction hierarchies," *Journal of Experimental Algorithmics (JEA)*, vol. 21, pp. 1–49, 2016.

[201] A. Schneck and K. Nökel, "Accelerating traffic assignment with customizable contraction hierarchies," *Transportation research record*, vol. 2674, no. 1, pp. 188–196, 2020.

[202] D. Van Vliet, *Saturn: Version 11.3 manual*, Apr. 2015. [Online]. Available: https://saturnsoftware2.co.uk/uploads/files/11.3.03G-Release-Notes.pdf.

[203] D. I. Robertson, ""TRANSYT" method for area traffic control," *Traffic engineering and control*, vol. 11, 1969.

[204] P. I. Richards, "Shock waves on the highway," *Operations Research*, vol. 4, no. 1, pp. 42–51, 1956, ISSN: 0030364X, 15265463. [Online]. Available: http://www.jstor.org/stable/167515.

[205] C. F. Daganzo, "A finite difference approximation of the kinematic wave model of traffic flow," *Transportation Research Part B: Methodological*, vol. 29, no. 4, pp. 261–276, 1995.

[206] P. Liu, X. Xu, A. Chen, C. Yang, and L. Xiao, "A select link analysis method based on logit–weibit hybrid model," *Journal of Modern Transportation*, vol. 25, no. 4, pp. 205–217, 2017.

[207] D. Galvin, "The independent set sequence of regular bipartite graphs," *Discrete Mathematics*, vol. 312, no. 19, pp. 2881–2892, 2012.

[208] U. Meyer and P. Sanders, "Delta-stepping: A parallel single source shortest path algorithm," in *European symposium on algorithms*, Springer, 1998, pp. 393–404.

[209] Y. Dinitz and R. Itzhak, "Hybrid bellman–ford–dijkstra algorithm," *Journal of Discrete Algorithms*, vol. 42, pp. 35–44, Jan. 2017. DOI: 10.1016/j.jda.2017.01.001. [Online]. Available: https://doi.org/10.1016/j.jda.2017.01.001.

[210] K. Wei, Y. Gao, W. Zhang, and S. Lin, "A modified dijkstra's algorithm for solving the problem of finding the maximum load path," in *2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT)*, IEEE, 2019, pp. 10–13.

[211] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders, "A parallelization of dijkstra's shortest path algorithm," in *International Symposium on Mathematical Foundations of Computer Science*, Springer, 1998, pp. 722–731.

[212] U. Pape, "Implementation and efficiency of moore-algorithms for the shortest route problem," *Mathematical Programming*, vol. 7, no. 1, pp. 212–222, Dec. 1974. DOI: 10.1007/bf01585517. [Online]. Available: https://doi.org/10.1007/bf01585517.

[213] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958, ISSN: 0033569X, 15524485. [Online]. Available: http://www.jstor.org/stable/43634538.

[214] S. Pettie, "A new approach to all-pairs shortest paths on real-weighted graphs," *Theoretical Computer Science*, vol. 312, no. 1, pp. 47–74, 2004.

[215] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[216] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, vol. 34, no. 3, pp. 596–615, Jul. 1987, ISSN: 0004-5411. DOI: 10.1145/28869.28874. [Online]. Available: https://doi.org/10.1145/28869.28874.

[217] R. B. Dial, "Algorithm 360: Shortest-path forest with topological ordering [h]," *Commun. ACM*, vol. 12, no. 11, pp. 632–633, Nov. 1969, ISSN: 0001-0782. DOI: 10.1145/363269.363610. [Online]. Available: https://doi.org/10.1145/363269.363610.

[218] L. Arge, L. Toma, and N. Zeh, "I/o-efficient topological sorting of planar dags," in *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, 2003, pp. 85–93.

[219] A. Kershenbaum, "A note on finding shortest path trees," *Networks*, vol. 11, no. 4, pp. 399–400, 1981.

[220] D. P. Bertsekas, "A simple and fast label correcting algorithm for shortest paths," *Networks*, vol. 23, no. 8, pp. 703–709, Dec. 1993. DOI: 10.1002/net.3230230808. [Online]. Available: https://doi.org/10.1002/net.3230230808.

[221] F. B. Zhan and C. E. Noon, "Shortest path algorithms: An evaluation using real road networks," *Transportation science*, vol. 32, no. 1, pp. 65–73, 1998.

[222] L. R. Ford Jr, "Network flow theory," Rand Corp Santa Monica Ca, Tech. Rep., 1956.

[223] M. Głkabowski, B. Musznicki, P. Nowak, and P. Zwierzykowski, "Efficiency evaluation of shortest path algorithms," in *AICT 2013, The Ninth Advanced International Conference on Telecommunications*, 2013, pp. 154–160.

[224] M. J. Bannister and D. Eppstein, "Randomized speedup of the bellman–ford algorithm," in *2012 Proceedings of the Ninth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, SIAM, 2012, pp. 41–47.

[225] A. Davidson, S. Baxter, M. Garland, and J. D. Owens, "Work-efficient parallel gpu methods for single-source shortest paths," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, IEEE, 2014, pp. 349–359.

[226] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A high-performance graph processing library on the gpu," in *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ACM, 2016, p. 11.

[227] Y. Wang, "High-performance graph primitives on the gpu: Design and implementation of gunrock," in *GPU Technology Conference*, 2014.

[228] F. Khorasani, K. Vora, R. Gupta, and L. N. Bhuyan, "CuSha: Vertex-centric graph processing on GPUs," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, ACM, 2014, pp. 239–252.

[229] NVIDIA, *Nvgraph*, Last accessed 2016-08-31, 2016. [Online]. Available: https://developer.nvidia.com/nvgraph.

[230] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, *Boost Graph Library: User Guide and Reference Manual, The*. Pearson Education, 2001.

[231] D. Gregor and A. Lumsdaine, "The parallel BGL: A generic library for distributed graph computations," in *In Parallel Object-Oriented Scientific Computing (POOSC*, vol. 2, Citeseer, 2005, pp. 1–18.

[232] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, Jun. 1962, ISSN: 0001-0782. DOI: 10.1145/367766.368168. [Online]. Available: https://doi.org/10.1145/367766.368168.

[233] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. ACM*, vol. 24, no. 1, pp. 1–13, Jan. 1977, ISSN: 0004-5411. DOI: 10.1145/321992.321993. [Online]. Available: https://doi.org/10.1145/321992.321993.

[234] PTV AG, *PTV VISUM 17 new features at a glance*, 2017. [Online]. Available: https://www.ptvgroup.com/fileadmin/user_upload/Products/PTV_Visum/Documents/Brochures_Features/PTV-Visum17-Features-EN.PDF.

[235] D. Van Vliet, "Saturn-a modern assignment model," *Traffic Engineering & Control*, vol. 23, no. HS-034 256, 1982.

[236] J. Dibbelt, B. Strasser, and D. Wagner, "Fast exact shortest path and distance queries on road networks with parametrized costs," in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2015, pp. 1–4.

[237] PTV America, *Ptv system requirements*, Last Accessed 2021-01-20, Nov. 14, 2019. [Online]. Available: https://company.ptvgroup.com/en-us/system-requirements.

[238] I. Wright, Y. Xiang, L. Waller, J. Cross, E. Norton, and D. Van Vliet, "The practical benefits of the saturn origin-based assignment algorithm and network aggregation techniques," in *European Transport Conference, 2010, Association for European Transport (AET)*, Citeseer, 2010.

[239] A. Cox, "Highways England's Regional Traffic Models challenges over the past year," in *SATURN User Group Annual Meeting November 2016*, Nov. 2, 2016.

[240] Z. Shen, K. Wang, and F. Zhu, "Agent-based traffic simulation and traffic signal timing optimization with gpu," in *Intelligent transportation systems (itsc), 2011 14th international ieee conference on*, IEEE, 2011, pp. 145–150.

[241] P. AG, *VISSIM 5.40 User Manual.* epubli GmbH, 2012, ISBN: 9783844214017. [Online]. Available: https://books.google.co.uk/books?id=trNX3HiAC6YC.

[242] Transport Simulation Systems, *About aimsun webpage*, Accessed: 2016. [Online]. Available: https://www.aimsun.com/aimsun/ (visited on 10/12/2016).

[243] Transport Simulation Systems, *Aimsun 8 dynamic simulators users' manual*, 2014.

[244] Roads Task Force, "Technical note 10 - what is the capacity of the road network for private motorised traffic and how has this changed over time," Transport for London, Tech. Rep., 2013. [Online]. Available: http://content.tfl.gov.uk/technical-note-10-what-is-the-capacity-of-the-road-network-for-private-motorised-traffic.pdf.

[245] E. Hermellin and F. Michel, "Complex flocking dynamics without global stimulus," in *Artificial Life Conference Proceedings 14*, MIT Press, 2017, pp. 513–520.

[246] S. Green, "Particle simulation using cuda," *NVIDIA whitepaper*, vol. 6, pp. 121–128, 2010. [Online]. Available: https://developer.download.nvidia.com/assets/cuda/files/particles.pdf.

[247] Z. Hao, R. Boel, and Z. Li, "Model based urban traffic control, part ii: Coordinated model predictive controllers," *Transportation Research Part C: Emerging Technologies*, vol. 97, pp. 23–44, 2018.

[248] J. R. D. Frejo, I. Papamichail, M. Papageorgiou, and B. De Schutter, "Macroscopic modeling of variable speed limits on freeways," *Transportation Research Part C: Emerging Technologies*, vol. 100, pp. 15–33, 2019.

[249] R. Bradley, I. Wright, D. Swain, *et al.*, "Accelerating traffic models using GPU-based technology," in *European Transport Conference 2016Association for European Transport (AET)*, 2016.

[250] D. Merrill, M. Garland, and A. Grimshaw, "Scalable gpu graph traversal," *ACM Sigplan Notices*, vol. 47, no. 8, pp. 117–128, 2012.

[251] Silverfrost Limited, *Silverfrost ftn95 user guide*, 2019. [Online]. Available: https://www.silverfrost.com/ftn95-help/ftn95_contents.aspx (visited on 01/10/2020).

[252] Intel Corporation, *Intel® fortran compiler release notes*, 2019. [Online]. Available: https://software.intel.com/en-us/articles/intel-fortran-compiler-release-notes.

[253] Transport for London, *London's strategic transport models*, Last Accessed 2021-01-26, Nov. 27, 2020. [Online]. Available: http://content.tfl.gov.uk/londons-strategic-transport-models.pdf.

[254] H. Nguyen and Transport for London, "Sub-regional highway modelling in london using saturn," in *SATURN User Group Annual Meeting November 2012*, Nov. 22, 2012.

[255] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA tesla: A unified graphics and computing architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008. DOI: 10.1109/MM.2008.31.

[256] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, *Dissecting the nvidia volta gpu architecture via microbenchmarking*, 2018. arXiv: 1804.06826 [cs.DC]. [Online]. Available: https://arxiv.org/abs/1804.06826.

[257] Transport for London, *Sub-regional highway assignment model: Guidance on model use*, Sep. 2012. [Online]. Available: https://saturnsoftware2.co.uk/uploads/files/SATUGM16-Highways-England-Regional-Models-Update.pdf.

[258] O. Feldman, "The GEH measure and quality of the highway assignment models," in *European Transport Conference 2012Association for European Transport (AET) Transportation Research Board*, 2012. [Online]. Available: https://trid.trb.org/view/1324899.

[259] Highways England, *Design manual for roads and bridges (DMRB)*, Last Accessed 2021-01-12, Jan. 12, 2021. [Online]. Available: `https://www.standardsforhighways.co.uk/dmrb/`.

[260] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, ACM, 1967, pp. 483–485.

[261] E. Thonhofer and S. Jakubek, "Investigation of stochastic variation of parameters for a macroscopic traffic model," *Journal of Intelligent Transportation Systems*, vol. 22, no. 6, pp. 547–564, 2018.

[262] A. Gregoriades, "Towards a user-centred road safety management method based on road traffic simulation," in *2007 Winter Simulation Conference*, IEEE, 2007, pp. 1905–1914.

[263] P. Goswami, P. Schlegel, B. Solenthaler, and R. Pajarola, "Interactive sph simulation and rendering on the gpu," in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '10, Madrid, Spain: Eurographics Association, 2010, pp. 55–64.

[264] H. Sun, Y. Tian, Y. Zhang, *et al.*, "A special sorting method for neighbor search procedure in smoothed particle hydrodynamics on gpus," in *2015 44th International Conference on Parallel Processing Workshops*, IEEE, 2015, pp. 81–85.

[265] R. Chisholm, S. Maddock, and P. Richmond, "Improved gpu near neighbours performance for multi-agent simulations," *Journal of Parallel and Distributed Computing*, vol. 137, pp. 53–64, 2020.

[266] NVIDIA, "NVIDIA A100 tensor core GPU architecture," NVIDIA, Tech. Rep., 2020. [Online]. Available: `https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf`.

[267] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter, "Nvidia tensor core programmability, performance & precision," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, IEEE, 2018, pp. 522–531.

[268] M. J. Abraham, T. Murtola, R. Schulz, *et al.*, "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers," *SoftwareX*, vol. 1-2, pp. 19–25, 2015, ISSN: 2352-7110. DOI: `https://doi.org/10.1016/j.softx.2015.06.001`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S2352711015000059`.

[269] S. Le Grand, A. W. Götz, and R. C. Walker, "SPFP: Speed without compromise—a mixed precision model for GPU accelerated molecular dynamics simulations," *Computer Physics Communications*, vol. 184, no. 2, pp. 374–380, 2013, ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2012.09.022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465512003098.