Exploring Execution Schemes for Agent-Based Traffic Simulation on Heterogeneous Hardware

Jiajian Xiao
TUMCREATE and
Technische Universität München
jiajian.xiao@tum-create.edu.sg

Philipp Andelfinger
TUMCREATE and
Nanyang Technological University
pandelfinger@ntu.edu.sg

David Eckhoff
TUMCREATE and
Technische Universität München
david.eckhoff@tum-create.edu.sg

Wentong Cai
Nanyang Technological University
Singapore
aswtcai@ntu.edu.sg

Alois Knoll
Technische Universität München and
Nanyang Technological University
knoll@in.tum.de

***Abstract*—Microscopic traffic simulation is associated with substantial runtimes, limiting the feasibility of large-scale evaluation of traffic scenarios. Even though today heterogeneous hardware comprised of CPUs, graphics processing units (GPUs) and fused CPU-GPU devices is inexpensive and widely available, common traffic simulators still rely purely on CPU-based execution, leaving substantial acceleration potentials untapped. A number of existing works have considered the execution of traffic simulations on accelerators, but have relied on simplified models of road networks and driver behaviour tailored to the given hardware platform. Thus, the existing approaches cannot directly benefit from the vast body of research on the validity of common traffic simulation models. In this paper, we explore the performance gains achievable through the use of heterogeneous hardware when relying on typical traffic simulation models used in CPU-based simulators. We propose a partial offloading approach that relies either on a dedicated GPU or a fused CPU-GPU device. Further, we present a traffic simulation running fully on a many-core GPU and discuss the challenges of this approach. Our results show that a CPU-based parallelisation closely approaches the results of partial offloading, while full offloading substantially outperforms the other approaches. We achieve a speedup of up to 28.7x over the sequential execution on a CPU.**

I. Introduction

Agent-based microscopic traffic simulation has become an important tool for researchers and decision makers to investigate traffic-related phenomena and adjust traffic-control strategies [1]. In microscopic traffic simulation, each vehicle is an agent that determines its movement autonomously based on its surroundings. This level of detail is often needed to better understand traffic situations and to observe specific effects on the overall traffic system [2]. However, due to the increasing complexity of microscopic traffic simulation models, the microscopic approach can be associated with enormous computational costs, limiting its scalability.

Existing efforts to speed up agent-based traffic simulations can be categorised by the considered hardware platforms. Firstly, high-performance computing environments can be employed to distribute the workload to a large number of CPU-based compute nodes interconnected using a low-latency and high-throughput interconnect such as InfiniBand. Commonly, libraries such as OpenMP and MPI are used to parallelise across CPU cores and compute nodes.

The second class of approaches targets compute nodes equipped with accelerators such as GPUs, many-core CPUs, or Field-Programmable Gate Arrays (FPGAs). Increasingly, these types of accelerators are available even in commodity workstations. Segments of the simulation model code that are suitable to be offloaded to an accelerator, e.g., numerically intensive computations, are identified and ported to the target platform [3], [4]. By applying this approach, performance gains can be achieved while still allowing modellers to easily adapt the representation of the road network, statistics collection, input/output, and so forth. The frequent data transfers required by partial offloading approaches can be avoided by offloading the entire simulation to the accelerator [5], [6]. However, since code running on current accelerators must still be optimised for the given hardware to achieve highest performance, porting large segments of an existing simulator to an accelerator requires substantial development efforts and decreases the maintainability of the model code. Thus, there is a tradeoff between the performance gains on one hand, and the development efforts and maintainability on the other hand.

In this paper, we explore the performance benefits of execution schemes to exploit the computational capacities of dedicated CPUs, GPUs and fused CPU-GPU devices (Accelerated Processing Units, APUs). We consider a traffic simulation with a graph-based representation of the road network, and well-established models for car-following and lane-changing behaviour used in common CPU-based traffic simulators. The same graph-based road network representation and models are considered by all execution schemes to support simulationists in parallelisation decisions without the need to rely on hardware-specific models.

We first systematise the options for parallelisation based on the data dependencies defined by the Sense-Think-Act

978-1-5386-5048-6/18/$31.00 ©2018 IEEE

cycle of underlying agent-based models. For a resulting set of execution schemes employing many-core hardware, we discuss the required changes to an existing simulator architecture and assess the performance gains compared to an optimised CPU-based execution. We propose a simple partial offloading scheme relying either on a dedicated GPU or on an APU, achieving a speedup of up to 1.7x. A fully GPU-based execution scheme is shown to achieve a speedup of up to 28.7x at the cost of introducing complexity in the model development. Our implementations are made available to the community[1].

Our main contributions are as follows:

- We present the first comprehensive study on execution schemes for road traffic simulation on a heterogeneous CPU/GPU platform based on common road network and driver behaviour models.
- We propose partial and fully GPU-based execution schemes and describe the required modifications to the base case of a CPU-based traffic simulator.
- We present performance measurements comparing the execution schemes to the purely CPU-based execution.

The reminder of this paper is organised as follows: In Section II, we give an overview of related work. In Section III, we describe the set of models that constitute the considered agent-based traffic simulation. In Section IV, we explore different execution schemes for agent-based traffic simulation on heterogeneous hardware. In Section V, we discuss challenges and future work. Section VI concludes the paper.

II. Background and Related Work

In the following, we first outline the current hardware and software support for general-purpose computations on GPUs. Subsequently, we sketch basic concepts of agent-based simulation, which forms the basis for microscopic traffic simulation. Finally, we give an overview of existing work on executing agent-based traffic simulation using heterogeneous hardware.

A. General-Purpose Computing on GPUs

Due to the massively parallel architecture of Graphics Processing Units (GPUs), highly data-parallel tasks can be executed at high performance. However, most GPUs connect to the host CPU through a PCI-E bus. Thus, interactions between the host CPU and the GPU require data transfers between the CPU and GPU memory, the cost of which may reduce the benefits of GPU-based parallel processing. In recent years, architectures that fuse a CPU and a GPU on a single die have been introduced (cf. Figure 1). Since in these architectures, the CPU and GPU components are able to access the same memory, explicit data transfers can be avoided entirely. However, existing Accelerated Processing Unit (APU) products such as recent Intel and AMD processors have focused more on energy efficiency than performance. Thus, the computational power of the GPU portion compared to dedicated graphics cards is relatively low.

[1]https://github.com/xjjex1990/oclTrafficSimulator

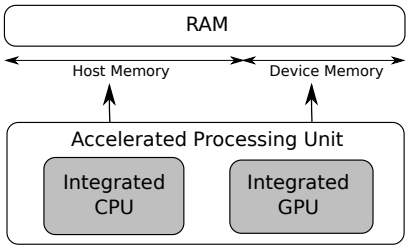


Fig. 1: The high-level architecture of an Accelerated Processing Unit. The CPU and the GPU both have direct access to main memory.

When accessing the memory of a dedicated GPU, the architecture prescribes a preference for memory access patterns where neighbouring threads access neighbouring memory locations. Such accesses can be *coalesced*, i.e., merged to reduce the number of memory transactions, increasing the effective memory throughput compared to scattered accesses.

Frameworks such as CUDA [7] and OpenCL [8] simplify the development of GPGPU code by enabling developers to program in C-like languages without considering low-level details of the hardware. CUDA is a proprietary development platform targeting NVIDIA graphics cards. OpenCL is an open standard for a programming framework targeting various computing devices such as CPUs, GPUs, and FPGAs, with implementations from a wide range of vendors.

In this paper, we will focus on OpenCL to run simulation code on heterogeneous hardware. Development of an OpenCL program involves API calls to allocate memory, to perform data transfer between the host and other devices, and calls to execute device code formulated in so-called *kernels*, which are executed in parallel by a configurable number of threads.

In OpenCL, threads are called *workitems*. A configurable number of workitems form a *workgroup*. Workitems in one workgroup have access to a certain amount of shared memory and can be synchronised efficiently.

B. Agent-Based Traffic Simulation

In agent-based simulations (ABS), the simulated entities perform actions based on the state of other entities and the simulated environment. Conceptually, an ABS typically follows a *Sense-Think-Act* cycle [9]: In the *Sense* stage, each agent detects its neighbours and gathers information from its environment. In the *Think* Stage, each agent makes decisions based on the information collected in the *Sense* stage. In the *Act* stage, each agent updates its state based on the decisions made in the Think stage. After a Sense-Think-Act cycle is completed, simulation time advances, and a new cycle starts.

In most existing agent-based traffic simulators such as SUMO [10], VISSIM [11] or CityMoS [12], vehicles are so-called Driver-Vehicle-Units (DVUs), composed of driver models and vehicle models. Driver models include models to describe car-following and lane-changing behaviour. Vehicle models represent the physical characteristics of individual vehicles as well as simulating components such as engines and auxiliary power consumers.

C. Traffic Simulation using Heterogeneous Hardware

Due to the opportunity to separate the computations according to the Sense-Think-Act cycle and the independent computations within each stage of the cycle, conceptually, agent-based traffic simulation (ABTS) lends itself to acceleration on many-core hardware. A survey of techniques for general agent-based simulation on heterogeneous hardware is given in [13]. Considering the existing works on ABTS using many-core devices, two general approaches can be differentiated: offloading approaches using a host CPU and a many-core device, and purely GPU-based approaches.

While offloading approaches have been widely explored in the context of discrete-event simulation [14], [15], most existing works on traffic simulation using many-core devices have focused on purely GPU-based execution. In purely GPU-based approaches, the entire simulation is executed on the GPU so that significant communication with the host CPU is only required at the start and end of the simulation. However, this approach makes it necessary to adapt the data structures and the control flow to the hardware properties of the GPU. Further, debugging and extending the simulator may require expert knowledge in parallel computing and the consideration of hardware-specific details.

Perumalla [5] proposes to map a graph-based network onto a grid in a GPU-based traffic simulation. Since this representation is highly suited to the GPUs architecture, the approach enables simulations at the scale of road networks covering entire states of the USA. The considered model is field-based, i.e., vehicles probabilistically move in a certain direction at each cell in the grid, each cell storing the number of agents currently residing at the cell.

In the approach proposed by Strippgen and Nagel [16], each road is represented by a first-in, first-out queue stored as a ring buffer, one GPU thread processing one road. Since a vehicle's mobility to the end of each road is determined directly from the speed limit and road length, their simulation can be considered mesoscopic instead of microscopic.

Hirabayashi et al. [3] compare two approaches to purely GPU-based ABTS on a single-lane road based on the Optimal Velocity model [17]: as in most other works, in the first approach, the CPU calls GPU kernels to execute the agent updates at each step in model time. In the second approach, the entire simulation is performed within a single kernel call. Since synchronisation across thread blocks is not supported within a kernel, the authors quantify the error incurred by the lack of synchronisation.

Wang et al. [18] execute road traffic simulations of an infinite-length two-lane road on a dedicated GPU or the GPU portion of an APU. The main focus of their work is an efficient neighbour discovery algorithm on an APU. While all main parts of the simulation run on the GPU, a merging step required when agents enter a lane can be performed on the CPU. As in our work, the Intelligent Driver Model [19] is employed to simulate car-following. For lane-changing, authors rely on the MOBIL model [20]. Of the existing works, this is the closest to our present paper, since it shares our intention to compare execution approaches on heterogeneous hardware and employs common driver behaviour models. The main difference to our work is the reliance on a single road instead of a road network. Thus, Wang et al. rely on bulk GPU operations on two large arrays holding all vehicles, whereas our simulation using a graph-based road network requires fine-grained operations on hundreds of thousands of small arrays representing one lane each.

Heywood et al. [6] use their FLAME GPU framework to execute a traffic simulation running entirely on a GPU. Vehicles accelerate according to Gipps' car-following model [21] on grid road networks. Neighbouring agents are not stored in a joint data structure associated with each lane but communicate each update in position and velocity using a messaging system. In contrast to our work, their focus lies more on different messaging systems than on exploring the possibilities for offloading to many-core devices.

Finally, an approach to accelerate parameter studies is proposed by Shen et al. [22]. They consider simulations of a small road network of six intersections, executing multiple replications of the traffic simulation in parallel on a GPU. Their work relies on the GM car-following model [23], but disregards lane-changing. The focus is on exploring the possibility to execute large numbers of simulations in parallel.

While many options for GPU-based simulations have been investigated by the existing works, we are not aware of any previous work evaluating offloading opportunities under the constraint of maintaining a graph-based road network representation and well-known models for car-following and lane-changing as in common CPU-based simulators. Thus, our results can support simulationists in the parallelisation of their simulation systems without the need to rely on models tailored to the GPU platform.

III. Simulation Models and System

We consider a traffic simulation running on an arbitrary road network represented as a graph. Edges represent road segments, nodes represent intersections. Each road segment may have multiple lanes.

The driver behaviour is governed by two essential models: a car-following model and a lane-changing model.

Each car follows the vehicle ahead and avoids collisions by adjusting its acceleration and deceleration at every time step. For car-following behaviour, we employ the Intelligent Driver Model (IDM) [19]. The input parameters of IDM include the current velocity, a desired velocity, and the distance and speed of the vehicle in front. IDM returns the acceleration for the next time step limited by a maximum possible acceleration.

While the car-following model controls the longitudinal movement of the vehicle, lateral movement is handled by the lane-changing model. Typically, the simulated vehicle evaluates whether changing the lane could allow it to accelerate (Discretionary Lane Change, DLC) or whether a lane change is necessary to continue its route (Mandatory Lane Change, MLC). In this paper, we use Ahmed's lane-changing model

TABLE I: Average and peak number of agents on the road depending on the agent generation rate.

Generation Rate	Average	Peak
2	4 557	6 257
5	7 257	20 745
10	9 788	35 423
20	11 291	44 922
50	12 218	48 964
100	12 717	49 652

[24], [25]. In Ahmed's model, the DLC decision is made based on evaluating whether turning into the gap bounded by the preceding and succeeding vehicles in the neighbouring lane improves a utility function affected by the difference to the desired velocity and the gaps on the current and neighbouring lanes. A decision is made based on two random numbers to determine probabilistically whether the lane-changing manoeuvre is actually performed.

The traffic intensity is varied by configuring the *generation rate*, which is the number of agents generated at each time step of 250ms, creating different levels of congestion (cf. Table I). In total, 50 000 agents are generated.

The considered traffic simulation is executed on a model of the road network of the city-state of Singapore. The agents' routes are pre-calculated based on the shortest path given a origin-destination pair drawn uniformly at random. The performance measurements are performed on a system equipped with an Intel Core i7-4770, 16 GB of RAM and a dedicated NVIDIA GTX 1060 graphics card with 6 GB of RAM. An execution scheme for partial offloading is evaluated on an APU platform with a dual-core Intel Core i5-4278U, an integrated Intel Iris Graphics 5100 GPU and 16 GB of RAM.

IV. Execution Schemes

A. Overview

As a starting point, we consider a CPU-based sequential microscopic traffic simulation as implemented in common simulation frameworks such as Repast [26] and Mason [27]

Each vehicle is represented by an agent, whose behaviour is determined by several models, each following the Sense-Think-Act cycle. Pseudo code of the core simulation loop is given in Algorithm 1.

To identify potentials for parallelisation, we consider the dependencies among the stages and within each stage (cf. top of Figure 2). We can observe that the Sense and Think stages are independent across agents and are thus candidates for parallelisation. During the Act stage, agents may affect their neighbours, e.g., by attempting to enter the same position on a lane as another agent. Since these interactions are difficult to predict, parallelisation of the Act stage is non-trivial. Thus, in our CPU-based execution scheme (OMP-SENSE-THINK-CPU), used as a baseline in our experiments, we parallelise only the Sense and Think stages.

Considering opportunities for offloading the stages to a GPU, we note that for each agent, the Sense stage requires access to a portion of the road network and agent states to gather the relevant neighbour states. In effect, the Sense stage requires access to most or all of the simulation state and is thus not well-suited for offloading. Similarly, updating the agent states in the Act stage requires access to most or all of the agent states. Thus, we propose two variants of an offloading scheme that executes the Think stage on a GPU: in OCL-THINK-GPU, we execute the Think stage on a dedicated GPU, which requires data transfers over the PCI-E bus before and after the computations. In OCL-THINK-AGPU, we offload to the GPU portion of an APU. Although the limited compute resources of the integrated GPU constrain the performance benefits in terms of pure computation, the shared access to main memory by the CPU and the GPU portion allows us to eliminate data transfer delays.

```
while termination criterion not satisfied do
    foreach agent do
        agent.model1.sense()
        agent.model1.think()
        agent.model1.act()
        agent.model2.sense()
        agent.model2.think()
        agent.model2.act()
        ...
    end
    advance simulation time
end
```

Algorithm 1: Pseudo code of the core execution loop of an agent-based simulation. The agent interactions during the Act stage prohibit trivial parallelisation of the loop iterations.

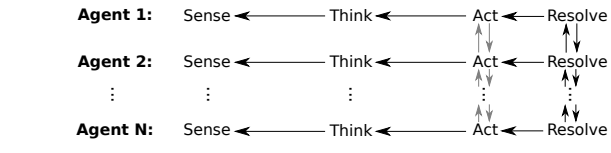
Finally, we explore a fully GPU-based execution scheme (OCL-ALL-GPU) in which all stages are executed on a dedicated GPU. This scheme eliminates all major data transfers during the main simulation loop but requires porting the entire simulator engine to the GPU. Since properties of a graph-based road network representation are exploited, OCL-ALL-GPU is specific to the considered models, whereas the other execution schemes are applicable to other agent-based models that follow a Sense-Think-Act cycle.

Since our OpenCL implementation also allows for execution on a CPU, we compare the execution schemes with the same implementations running in parallel on a CPU (OCL-THINK-CPU), the CPU portion of an APU (OCL-THINK-ACPU), and a fully parallelised CPU-based execution (OCL-ALL-CPU).

The bottom of Figure 2 lists the execution schemes together with the means to carry out the required data transfers. In the remainder of this section, we describe each execution scheme in detail. For each scheme, we discuss the overall design, implementation concerns as well as the results of our performance measurements.

B. CPU-Based Execution

The starting point for our work is a sequential CPU-based traffic simulator as represented by common academic or commercial simulators such as SUMO or VISSIM. The simulation models were extracted from CityMoS, a CPU-based microscopic traffic simulator [12]. To acquire a fair baseline



***-CPU**	*CPU*	-	*CPU*	-	*CPU*	-	-
OCL-THINK-GPU	*CPU*	*PCI-E*	*GPU*	*PCI-E*	*CPU*	-	-
OCL-THINK-AGPU	*CPU*	*Zero-Copy*	*GPU*	*Zero-Copy*	*CPU*	-	-
OCL-ALL-GPU	*GPU*	-	*GPU*	-	*GPU*	-	*GPU*

Fig. 2: Top: dependencies among the stages in a simulation time step. If inter-agent dependencies are ignored during the Act stage, a separate conflict resolution stage is required. Bottom: the execution schemes considered in our experiments and the means of data transfer from one stage to the next.

```
while termination criterion not satisfied do
    foreach agent in parallel do
        agent.model1.sense()
        agent.model2.sense()
        ...
    end
    foreach agent in parallel do
        agent.model1.think()
        agent.model2.think()
        ...
    end
    foreach agent do
        agent.model1.act()
        agent.model2.act()
        ...
    end
    advance simulation time
end
```

Algorithm 2: Pseudo code of the core execution loop of an agent-based simulation after applying loop fission. Since the Sense and Think stages are independent across agents, the first two for-loops can both be trivially parallelised.

for performance comparisons, we parallelise the portions of the simulation that do not require substantial changes to an existing simulator architecture.

1) Architecture: In the CPU-based execution scheme, we execute the entire simulation on the CPU. As discussed above, the Sense and Think stages can be executed independently for each agent. Parallelisation of the Act stage mandates efficient synchronisation and conflict resolution and thus profound changes to the simulator architecture, which a variety of existing works have explored [28], [29]. Since our focus is on the transition from a CPU-based simulator to a heterogeneous execution, we restrict the parallelisation of our initial CPU-based implementation to the Sense and Think stage. A fully parallelised execution scheme will be explored in Section IV-D. In this work, we focus on parallelisation within a single execution node.

Considering the pseudo code of Algorithm 1, there is one loop iteration for each agent, each iteration covering all models for the current agent. Since the agent states are only updated in the Act stage, Sense and Think can be parallelised

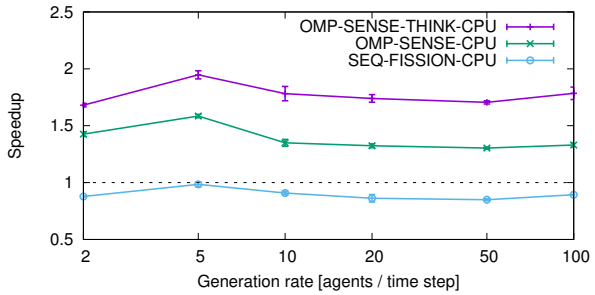


Fig. 3: Speedup with standard errors when parallelising Sense (OMP-SENSE-CPU) or Sense and Think (OMP-SENSE-THINK-CPU) over sequential execution (SEQ-CPU). Loop fission (SEQ-FISSION-CPU) results in a slight slowdown.

across all agents. However, the data dependencies given by the interactions among agents in the Act stage prohibit a straightforward parallelisation. Thus, we adapt the control flow to separate the Sense and Think stage from the Act stage.

The required transformation of splitting a loop into multiple loops is known as *loop fission* [30]. Algorithm 2 lists pseudo code of the simulation after loop fission. Although the computational steps are unchanged on a conceptual level, loop fission distributes the accesses to each agent's state variables across multiple loops. Due to the decrease in memory access locality, a performance decrease must be expected. We study the effect of loop fission on the performance in Section IV-B3.

2) Implementation: We restructured the simulator code by applying loop fission. The parallelisation is performed using OpenMP by annotating the for-loops of the Sense and Think stages. To limit contention for the workload among the CPU threads, we configured a chunk size of 1000 agents. During the Sense stage of the car-following and lane-changing models, the respective output is stored in a per-agent element of an array. The two resulting arrays form the input to the Think stage of the two models.

For lane-changing behaviour, we rely on Ahmed's model, which requires drawing two uniformly distributed random numbers in $[0, 1]$ for each agent at each time step. To achieve a fair comparison with the GPU-based implementations, we use the MWC64X generator[2], for which efficient implementations exist both in plain C and OpenCL.

3) Performance Evaluation: The experiment is conducted on the platform equipped with a dedicated GPU. As illustrated in Figure 3, loop fission slightly decreases performance. The runtime was increased by 15% in the worst case. When applying the OMP-SENSE-CPU scheme, i.e., with only the Sense stage parallelised by OpenMP, a performance gain of at least 1.30x is observed. The speedup is increased by also parallelising the Think stage (OMP-SENSE-THINK-CPU), achieving at least 1.68x for all cases. The highest speedup is achieved when the agent generation rate is 5 per time step for both the OMP-SENSE-CPU and OMP-SENSE-THINK-CPU schemes, with a respective speedup of 1.58x and 1.95x.

[2]http://cas.ee.ic.ac.uk/people/dt10/research/rngs-gpu-mwc64x.html

C. Partial Offloading

In the following, we aim at accelerating the simulation in a heterogeneous CPU/GPU environment. We first explore an offloading approach [31], [32], where computationally intensive portions of the simulation are offloaded to the GPU.

1) Architecture: The offloading approach follows the idea of the parallelised CPU-based execution: we exploit the independence of per-agent stages. However, since the GPU does not have direct access to the host memory, data transfers over the PCI-E bus are introduced. The Think stage is a natural candidate for offloading, since it relies only on the output of the Sense stage. In contrast to this, the Sense stage accesses both the static environment, i.e., the lengths and speed limits of nearby road segments, and the states of nearby agents. Similarly, an agent's Act stage relies on the static environment and may interfere with the agent's neighbours. Thus, offloading the Sense or Act stage would require transferring substantial parts of the simulation state to the GPU at each time step, which instead suggests porting the entire simulation to the GPU. Based on this reasoning, we offload only the Think stage and explore full offloading in Section IV-D. As in the CPU-based scheme, the Sense stage is executed in parallel on the CPU. The Act stage is executed sequentially on the CPU.

At the start of each Think stage, the CPU transfers the data needed for the computation from the host memory to the graphics memory. The GPU processes the data and transfers the results back to the host memory. Further, we explore the offloading scheme using an APU, allowing us to avoid data transfers over the PCI-E bus.

2) Implementation: The implementation of the offloading scheme follows the general approach used for the Think stage in the CPU-based scheme, replacing the for-loop of the Think stage with a call to GPU code that executes the stage in parallel for all agents. Our implementation is based on OpenCL, which allows us to execute the same code on a CPU, GPU, or APU. OpenCL API calls are used to transfer the model input data to the GPU, to call an OpenCL kernel executing the model, and to transfer the model output data back to the host memory. On the GPU, each thread executes the model for one agent. The OpenCL implementation of the models is nearly identical with the plain C++ implementation, apart from the indexing based on GPU threads required in OpenCL. Additional arrays in graphics memory are used to store the model input and output for each agent. The input array is filled by a data transfer from host memory prior to the OpenCL call. After the OpenCL call has finished, the content of the output array are transferred to host memory. On the CPU, each agent reads the respective output values from the output array during the subsequent Act stage. Quantitatively, the input data comprises 16 bytes per agent for the car-following model and 52 bytes for the lane-changing model. The output for each models comprises 4 bytes per agent. The implementation targeting APUs follows the same general process. However, to utilise the zero-copy technique, the input and output arrays accessed by the

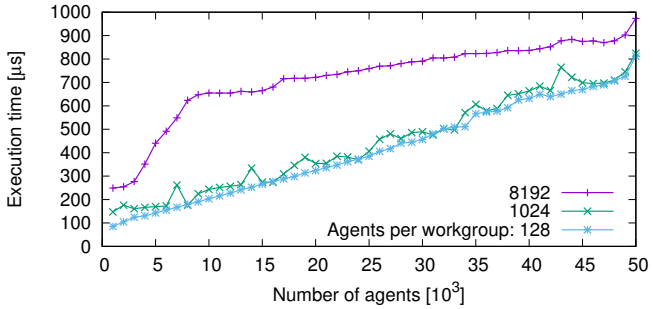


Fig. 4: Total execution time on the dedicated CPU for the Think stage of the car-following model and lane-changing model when varying the workgroup size.

OpenCL kernels are created using the OpenCL memory flag *CL_MEM_USE_HOST_PTR* [33], which allocates memory in a shared space that can be accessed by the CPU and the GPU portion, avoiding data transfers.

3) Performance Evaluation: We evaluate the partial offloading with respect to purely CPU-based execution and offloading of the Think stage to the GPU, both for the platform with a dedicated GPU (OCL-THINK-CPU and OCL-THINK-GPU) and for the APU platform (OCL-THINK-ACPU and OCL-THINK-AGPU).

The configured workgroup size in OpenCL can have a substantial impact on the overall performance. The best-performing workgroup size depends on properties of the hardware and the computation to be performed [34]. To find the best configuration, we vary the workgroup size from 128 to 8192 for the CPU and 128 to 1024 for the GPU both for the car-following and the lane-changing kernels. As depicted in Figure 4, the smallest workgroup size 128 always leads to best performance on the CPU. In contrast to the CPU, we observe that the workgroup size configurations we studied do not have an obvious impact on the GPU performance. We set the GPU's workgroup size to 128. The same value of 128 was identified to achieve the best performance on both the CPU and GPU portion of the APU platform. In the remainder of the paper, we will use the best configurations in all measurements.

Figure 5 shows a comparison of partial offloading execution schemes. A speedup of 1.8x over sequential execution on the CPU (SEQ-CPU) is achieved for OCL-THINK-GPU, i.e., offloading the Think stage to the dedicated GPU. OCL-THINK-CPU produces a slightly better result, achieving a speedup up to 2.0x, due to the relatively lightweight computations in relation to the data transfer overheads introduced by OCL-THINK-GPU.

However, on the APU, on which the data transfer overhead for offloading is eliminated, OCL-THINK-AGPU achieves an overall speedup of up to 1.7x over CPU-SEQ on the CPU portion of the APU, outperforming both the OCL-THINK-ACPU and OMP-SENSE-THINK-ACPU schemes (cf. Figure 6).

Although the Think stage can be accelerated substantially by offloading, the overall performance gain is limited by Amdahl's law: since the Think stage constitutes about 18.6%

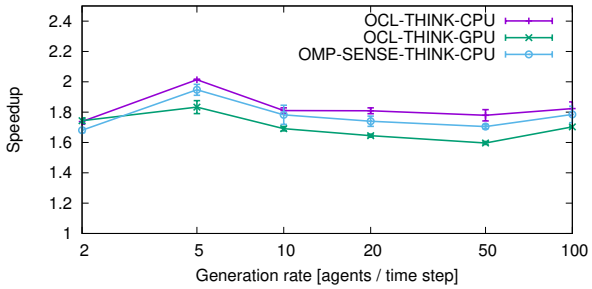


Fig. 5: Speedup with error bars showing standard errors over sequential execution when parallelising Sense by OpenMP and Think by OpenCL (OCL-THINK-CPU) and when offloading Think to the dedicated GPU (OCL-THINK-GPU).

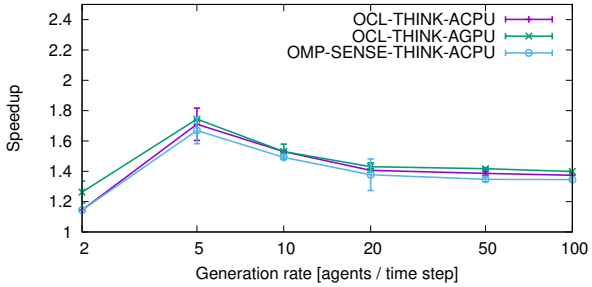


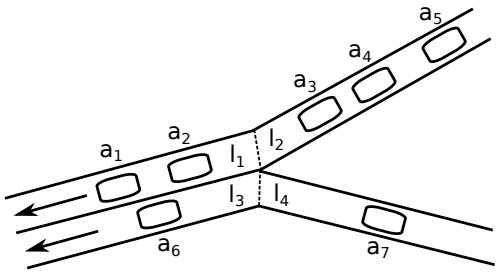
Fig. 6: Speedup with error bars showing standard errors over sequential execution when parallelising Sense by OpenMP and Think by OpenCL on the CPU portion of an APU (OCL-THINK-ACPU) and when offloading Think to the GPU portion of an APU (OCL-THINK-AGPU).

of the runtime on the APU platform given the Sense stage has already been parallelised, an upper bound on the speedup is given by $1/(1-0.186) = 1.23$. Thus, our aforementioned speedup of 1.18x is close to the theoretical maximum if offloading is limited to the Think stage.

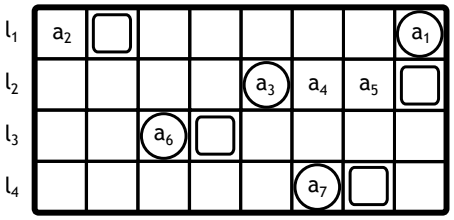
D. Full Offloading

In the previous section, we observed that offloading the Think stage can already give some performance gains. However, even if we avoid data transfers using zero-copy memory access on an APU, the performance gains are limited by Amdahl's law. As discussed in Section IV-C, offloading Sense and Act requires access to nearly all of the simulation state and static environment data. This situation suggests a full port of the simulator to OpenCL instead. In the following, we present and evaluate a fully offloaded traffic simulator running entirely on a many-core GPU. Due to the implementation in OpenCL, the simulator also supports a parallelised execution on a CPU.

1) Architecture: In this execution scheme, major data transfers are only required during initialization to set up the static environment and to provide the simulator with the initial scenario parameters. Subsequently, the simulation proceeds as a sequence of OpenCL kernel calls, with minor data



(a) Example of a portion of the graph-based road network, showing 3 links with a total of 4 lanes (l_1 to l_4) and 7 vehicles (a_1 to a_7).



(b) Representation of the road network in memory. Each lane l_i is a ring buffer holding vehicles a_j ordered by their position on the lane, with associated indexes for the head (circle) and the initial index of vehicles entering from other lanes (rectangles).

Fig. 7: Road network representation in OCL-ALL-CPU and OCL-ALL-GPU.

transfers to signal the termination of the simulation. Output of simulation statistics may be performed either by data transfers during the simulation or once the termination criterion is met.

Considering the dependencies between the agent stages, the Sense and Think stages are now both trivially parallelisable. However, when parallelising the Act stage, the potential interactions among agents must be considered: when moving, multiple agents may enter overlapping positions in the simulation space. While a sequential execution with one-by-one position updates can avoid such situations, a parallelised execution requires a conflict resolution mechanism to achieve consistent, i.e., collision-free, agent states. We perform conflict resolution after the Act stage.

2) Implementation: In our implementation of the fully GPU-based execution scheme, each agent is a structure comprised of a numerical identifier as well as the current and desired lane, position, and velocity. Each lane in the road network is represented by a ring buffer holding the agents currently on the lane, ordered by their positions (cf. Figure 7). As in the offloading version, apart from the indexes used to access the input and output data, the OpenCL code for the car-following and lane-changing model are nearly identical with the CPU implementation.

Six OpenCL kernels are called at each time step:

1. SpawnVehicles: in this kernel, a single thread creates and initialises a configurable number of new vehicles.

2. CollectActiveLanes: as preparation for the Sense and Think stages, lanes with at least one vehicle are gathered in an array. Each GPU thread evaluates one lane.

3. SenseAndThink: this kernel combines the Sense and Think stage of both the car-following and lane-changing model. Each workgroup operates on one active lane, with each thread handling one agent at a time.

4. Act: this kernel actualises the desired lane, position and velocity determined in SenseAndThink. Since the Act stage may affect the number and indexes of agents on each lane, we avoid synchronisation by executing only a single thread per lane. However, when an agent enters a lane, synchronisation is still needed to avoid inconsistencies when multiple agents enter the same lane at the same time step. To this end, an atomic operation increments the index of the insertion position at the tail of the target ring buffer (rectangles in Figure 7b), retrieving the old index. The agent can then safely be stored at the old index. The insertion into the sorted ring buffer is performed in the next kernel. We mark agents that leave the current lane so they can be removed by the next kernel.

5. SortLanes: during the Act stage, agents may change lanes, which may affect the relative positions of agents on the target lane. To restore the ordering on each lane, we sort the vehicles by position, including new agents entering the lane. Agents that leave a lane are removed from the ring buffer. Since each lane typically holds at most a few dozen agents, we apply sequential quicksort using one thread per lane.

6. ResolveConflicts: overlaps between the agents are resolved by moving a vehicle that has performed a lane change or advanced to the next link back to the original lane. If more than one vehicle involved in a conflict has entered a new link, the vehicle that is further behind is moved. Each conflict resolution round may affect the ordering at each lane, and may also create new conflicts. Thus, SortLanes and ResolveConflicts are executed until no further conflicts occur.

The implementation based on ring buffers and the synchronisation based on atomic operations closely resembles GPU-based discrete-event simulations, which have been shown to achieve high speedup over a CPU-based execution [35], [36]. Our approach to conflict resolution postpones the conflict resolution to after the Act stage and iterates until all conflicts have been resolved based on the relative position of agents. Alternative approaches to conflict resolution for agent-based simulations on GPUs, as well as considerations of determinism and bias, have been studied in [37].

Since the CPU-based and partially offloaded versions execute the Act stage sequentially, conflicts can be avoided entirely. However, our conflict resolution approach affects the results only marginally compared to SEQ-CPU, with deviations in average travel times smaller than 3% for all tested parameter combinations.

3) Performance Evaluation: Initial performance evaluation runs showed that the performance of OCL-ALL-CPU and OCL-ALL-GPU was only marginally affected by the workgroup size. The measurements described in the following were executed with a workgroup size of 64 for all kernels.

As shown in Figure 8, the speedup achieved when parallelising all stages on the CPU (OCL-ALL-CPU) is up to 6.7x. A maximum speedup of 28.7x is achieved for OCL-ALL-GPU at low traffic density (agent generation rate of 2). In a more congested traffic scenario with an agent generation rate of 100, the performance gain by the parallelised execution is counteracted by the increasing overhead for conflict resolution, leading to a smaller overall performance gain. However, even in the most congested scenario, a speedup of 14.8x is achieved.

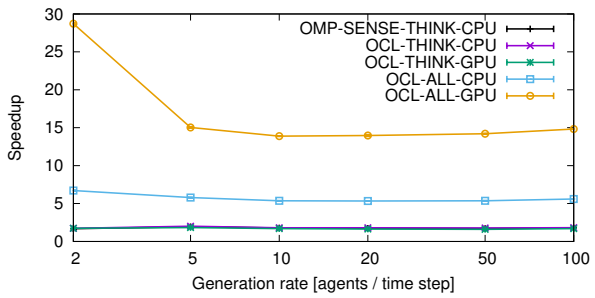


Fig. 8: Overall comparison of the execution schemes over sequential execution with error bars showing standard errors.

E. Comparison

Figure 8 gives an overview of the speedup gained by the different schemes on the platform with the dedicated GPU. Table II (generation rate = 2) and Table III (generation rate = 100) show the percentages of runtime per time step spent on the Sense, Think and Act stages. For the OCL-ALL-CPU and the OCL-ALL-GPU schemes, CollectActiveLanes and SenseAndThink are considered jointly as a single "Sense and Think" stage. The time spent on Act, SortLanes and ResolveConflicts is counted towards the Act stage.

OpenMP parallelisation in OMP-S-T-CPU (OMP-SENSE-THINK-CPU) halves the runtime for the Sense stage compared with SEQ-CPU, whereas the runtime spent on Think is reduced to less than a third. Offloading reduces the runtime spent on the Think stage by a factor of up to 16.

Due to the small contribution of the Think stage to the runtime, the speedup through partial offloading is modest, with a maximum speedup achieved through offloading is only up to 19%. On the hardware used in our experiments, the performance gains by parallelisation using OpenMP and OpenCL on the platform with the dedicated GPU are close (cf. Figure 8). Substantial speedup is achieved by the OCL-ALL-GPU scheme, i.e., executing the entire simulation on the GPU. The jointly considered Sense and Think stages are accelerated by a factor of 34.3 and 49.4 with an agent generation rate of 2 and 100, respectively. With the more congested scenario, a considerable amount of time is spent on the conflict resolution, with 79.6% spent on the Act stage.

V. Discussion

From the performance evaluation results, we can make a number of general observations: firstly, in the considered models, the Think stage takes up a relatively small portion of the overall runtime. Even in a scenario with a peak of about 50 000 vehicles concurrently on the road, Think only

TABLE II: Absolute and relative time spent on one iteration of each stage. Agent generation rate: 2 per time step.

	Absolute [ms]			Relative [%]		
	Sense	**Think**	**Act**	**Sense**	**Think**	**Act**
SEQ-CPU	7.94	2.41	4.92	52.0	15.8	32.2
OMP-S-T-CPU	3.64	0.71	4.74	40.0	7.8	52.2
OCL-THINK-CPU	3.69	0.20	4.90	42.0	2.3	55.7
OCL-THINK-GPU	3.78	0.39	4.58	43.2	4.5	52.3
OCL-ALL-CPU	0.60		1.67	26.5		73.5
OCL-ALL-GPU	0.32		0.21	59.8		40.2

TABLE III: Absolute and relative time spent on one iteration of each stage. Agent generation rate: 100 per time step.

	Absolute [ms]			Relative [%]		
	Sense	**Think**	**Act**	**Sense**	**Think**	**Act**
SEQ-CPU	15.29	5.95	9.77	49.3	19.2	31.5
OMP-S-T-CPU	6.65	1.58	9.14	38.3	9.1	52.6
OCL-THINK-CPU	7.03	0.41	9.58	41.3	2.4	56.3
OCL-THINK-GPU	7.37	0.53	10.31	40.5	2.9	56.6
OCL-ALL-CPU	1.50		4.05	27.0		73.0
OCL-ALL-GPU	0.43		1.66	20.4		79.6

contributes about 20% to the runtime. Hence, the gains achievable by offloading the Think stage are inherently limited: we achieved up to about 18% speedup by offloading to an APU, only slightly exceeding the speedup by a simple parallelisation of the Sense and Think stage on a CPU. For simplicity, a purely CPU-based parallelisation of these two stages may be preferable.

Secondly, substantial speedup can be achieved when parallelising all stages. The achieved runtime reductions by a factor of more than 28 on a GPU may put some large-scale parameter studies into reach that would be considered overly time-consuming using a sequential simulator. Since even on a multi-core CPU, our OpenCL implementation achieves a speedup of more than 6, our results show the suitability of common traffic simulation models for fine-grained parallelisation. Since only small amounts of computation are required at each time step of the simulation, retaining all simulation data within a single device contributes positively to the performance.

Our measurements are specific to the selected driver behaviour models: the Intelligent Driver Model for car-following, and Ahmed's model for lane-changing. However, we expect other microscopic traffic simulation models to exhibit similar demands in terms of input data, since a realistic representation of driver behaviour must be assumed to depend on the positions and velocities of nearby agents. Thus, for offloading approaches to achieve significant speedup even with the cost of the added data transfers, the computational demands of the models would need to be substantially larger than those of the models considered in the present paper. While the measurements cannot be generalised to other models, all considered execution schemes apart from OCL-ALL-GPU are applicable to other agent-based models that can be structured according to a Sense-Think-Act cycle. While OCL-ALL-GPU relies on the Sense-Think-Act cycle as well, it makes use of properties specific to traffic simulation on a graph-based road network and requires a model-specific conflict resolution step.

When considering the pure OpenCL simulator implementation, there are still obvious opportunities for runtime reduction. At high traffic density, more than three quarters of the simulation runtime are spent in the Act stage, which includes conflict resolution. Currently, each round of conflict resolution checks for conflicts with respect to every agent on every lane of the road network. However, rules could be formulated to limit the set of agents that may be involved in a conflict: for instance, depending on the considered models, conflicts may only result from agents either performing a lane change or advancing to the next lane. In such a case, considering only those agents that have entered a new lane may substantially reduce the cost of each round of conflict resolution.

An important challenge of a fully OpenCL-based parallelisation is the effect on the maintainability and extensibility of the simulator. Many common constructs are not easily available and will typically need to be substituted with implementations developed from scratch. Such constructs include the container types from the C++ standard template library, libraries for computing output statistics, facilities for disk input and output or for coupling with other simulation tools. This puts a substantial burden on the simulationist, who may not be an expert in parallel programming. An interesting avenue for future work may be a GPU-based traffic simulator core with an application programming interface that exposes common functionality for scenario generation, data aggregation, configuration of termination criteria, detection of simulation events, and so forth. With these functionalities, the simulation could be accelerated on a GPU, while still providing users with easy-to-use facilities for steering and analysing the simulation.

VI. Conclusions

In this paper, we explored execution schemes for partial and full offloading of microscopic traffic simulations to heterogeneous hardware. Our main findings are as follows: offloading of only parts of the simulation provides only minor benefits, since the most computationally demanding stages of each simulation time step require access to large portions of the simulation state. We showed that similar performance gains as with partial offloading can be more easily achieved by exploiting the concurrency of the trivially parallelisable parts of the simulation using a CPU. By executing the entire simulation on a GPU, data transfers are avoided, allowing for a speedup of up to 28.7x over a sequential CPU-based execution. However, the added need for resolving conflicts among agents as well as the higher complexity of the fully parallelised implementation may make simulation development more cumbersome. In future work, we intend to add generic facilities for common simulation tasks such as statistics aggregation and simulation coupling to the fully offloaded execution scheme. We aim for further performance gains by decreasing the overhead of conflict resolution. We hope that our publicly available implementation will support future studies in traffic simulation using heterogeneous hardware.

ACKNOWLEDGEMENT

This work was financially supported by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme.

REFERENCES

[1] P. A. Ehlert and L. J. Rothkrantz, "Microscopic Traffic Simulation with Reactive Driving Agents," in *Proceedings of the IEEE Intelligent Transportation Systems (ITSC '01)*. Oakland, CA, USA: IEEE, August 2001, pp. 860–865.

[2] W. Burghout, "Hybrid Microscopic-Mesoscopic Traffic Simulation," Ph.D. dissertation, KTH Royal Institute of Technology, 2004.

[3] M. Hirabayashi, S. Kato, M. Edahiro, and Y. Sugiyama, "Toward GPU-Accelerated Traffic Simulation and its Real-Time Challenge," in *Proceedings of the International Workshop on Real-time and Distributed Computing in Emerging Applications (REACTION '12)*. San Juan, Puerto Rico: Univ. Carlos III de Madrid, December 2012, pp. 45–50.

[4] R. Pavlov and J. P. Müller, "Multi-Agent Systems Meet GPU: Deploying Agent-Based Architectures on Graphics Processors," in *Proceedings of the Doctoral Conference on Computing, Electrical and Industrial Systems (DoCEIS '13)*. Costa de Caparica, Portugal: Springer, April 2013, pp. 115–122.

[5] K. S. Perumalla, B. G. Aaby, S. B. Yoginath, and S. K. Seal, "GPU-Based Real-Time Execution of Vehicular Mobility Models in Large-Scale Road Network Scenarios," in *Proceedings of the ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS '09)*. Lake Placid, NY, USA: IEEE, June 2009, pp. 95–103.

[6] P. Heywood, P. Richmond, and S. Maddock, "Road Network Simulation using FLAME GPU," in *Proceedings of the European Conference on Parallel Processing (Euro-Par '15)*. Vienna, Austria: Springer, August 2015, pp. 430–441.

[7] NVIDIA Corporation, *NVIDIA CUDA C Programming Guide*. Version 9.1.85, NVIDIA Corporation, 2018.

[8] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in science & engineering*, vol. 12, no. 3, pp. 66–73, May 2010.

[9] M. Hybinette, E. Kraemer, Y. Xiong, G. Matthews, and J. Ahmed, "Sassy: a design for a scalable agent-based simulation system using a distributed discrete event infrastructure," in *Proceedings of the Winter Simulation Conference (WSC '06)*. Monterey, CA, USA: IEEE, March 2006, pp. 926–933.

[10] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner, "SUMO (Simulation of Urban MObility)-an Open-Source Traffic Simulation," in *Proceedings of the Middle East Symposium on Simulation and Modelling (MESM '02)*. Sharjah, UAE: EUROSIS, October 2002, pp. 183–187.

[11] M. Fellendorf and P. Vortisch, *Microscopic Traffic Flow Simulator VISSIM*. Springer, June 2010, pp. 63–93.

[12] D. Zehe, S. Nair, A. Knoll, and D. Eckhoff, "Towards CityMoS: A Coupled City-Scale Mobility Simulation Framework," in *5th GI/ITG KuVS Fachgespräch Inter-Vehicle Communication*. Erlangen, Germany: FAU Erlangen-Nuremberg, April 2017.

[13] J. Xiao, P. Andelfinger, D. Eckhoff, W. Cai, and A. Knoll, "A Survey on Agent-Based Simulation using Hardware Accelerators," *arXiv preprint arXiv:1807.01014*, July 2018.

[14] D. W. Bauer, M. McMahon, and E. H. Page, "An Approach for the Effective Utilization of GP-GPUs in Parallel Combined Simulation," in *Proceedings of the Winter Simulation Conference (WSC '08)*. Miami, FL, USA: IEEE, December 2008, pp. 695–702.

[15] P. Andelfinger, J. Mittag, and H. Hartenstein, "GPU-Based Architectures and Their Benefit for Accurate and Efficient Wireless Network Simulations," in *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '11)*. Singapore: IEEE, July 2011, pp. 421–424.

[16] D. Strippgen and K. Nagel, "Using Common Graphics Hardware for Multi-Agent Traffic Simulation with CUDA," in *Proceedings of the International Conference on Simulation Tools and Techniques (Simutools '09)*. Rome, Italy: ICST, March 2009, pp. 62:1–62:8.

[17] M. Bando, K. Hasebe, K. Nakanishi, A. Nakayama, A. Shibata, and Y. Sugiyama, "Phenomenological Study of Dynamical Model of Traffic Flow," *Journal de Physique I*, vol. 5, no. 11, pp. 1389–1399, November 1995.

[18] J. Wang, N. Rubin, H. Wu, and S. Yalamanchili, "Accelerating Simulation of Agent-Based Models on Heterogeneous Architectures," in *Proceedings of the Workshop on General Purpose Processor Using Graphics Processing Units (GPGPU '06)*. Houston, TX, USA: ACM, March 2013, pp. 108–119.

[19] M. Treiber, A. Hennecke, and D. Helbing, "Congested Traffic States in Empirical Observations and Microscopic Simulations," *Physical review E*, vol. 62, no. 2, p. 1805, August 2000.

[20] M. Treiber and A. Kesting, "An Open-source Microscopic Traffic Simulator," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 3, pp. 6–13, December 2010.

[21] P. G. Gipps, "A Behavioural Car-Following Model for Computer Simulation," *Transportation Research Part B: Methodological*, vol. 15, no. 2, pp. 105–111, April 1981.

[22] Z. Shen, K. Wang, and F. Zhu, "Agent-Based Traffic Simulation and Traffic Signal Timing Optimization with GPU," in *Proceedings of the International Conference on Intelligent Transportation Systems (ITSC '11)*. Washington, DC, USA: IEEE, October 2011, pp. 145–150.

[23] X. Ma and I. Andréasson, "Driver Reaction Delay Estimation from Real Data and its Application in GM-type Model Evaluation," *Transportation Research Record*, no. 1965, pp. 130–141, December 2006.

[24] K. Ahmed, M. Ben-Akiva, H. Koutsopoulos, and R. Mishalani, "Models of Freeway Lane Changing and Gap Acceptance Behavior," in *Proceedings of the International Symposium on Transportation and Traffic Theory (ISTTT '96)*. Lyon, France: Elsevier, July 1996, pp. 501–515.

[25] K. I. Ahmed, "Modeling Drivers' Acceleration and Lane Changing Behavior," Ph.D. dissertation, Massachusetts Institute of Technology, 1999.

[26] N. Collier, "Repast: An Extensible Framework for Agent Simulation," *Natural Resources and Environmental*, vol. 8, p. Article No. 4, 2001.

[27] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "MASON: A Multiagent Simulation Environment," *SCSI Journal of Simulation*, vol. 81, no. 7, pp. 517–527, July 2005.

[28] S. El hadouaj, A. Drogoul, and S. Espié, "How to Combine Reactivity and Anticipation: The Case of Conflicts Resolution in a Simulated Road Traffic," in *Proceedings of the International Workshop on Multi-Agent Systems and Agent-Based Simulation (MABS '00)*, S. Moss and P. Davidsson, Eds. Boston, MA, USA: Springer, July 2000, pp. 82–96.

[29] Y. Xu, W. Cai, H. Aydt, M. Lees, and D. Zehe, "An Asynchronous Synchronization Strategy for Parallel Large-scale Agent-based Traffic Simulations," in *Proceedings of the Conference on Principles of Advanced Discrete Simulation (PADS '15)*. London, UK: ACM, June 2015, pp. 259–269.

[30] H. Zima and B. Chapman, *Supercompilers for Parallel and Vector Computers. ACM-Press Frontier Series*. Addison-Wesley, 1990.

[31] J. Liu, Y. Liu, Z. Du, and T. Li, "GPU-Assisted Hybrid Network Traffic Model," in *Proceedings of the Conference on Principles of Advanced Discrete Simulation (PADS '14)*. Denver, CO, USA: ACM, May 2014, pp. 63–74.

[32] R. M. Fujimoto, C. Carothers, A. Ferscha, D. Jefferson, M. Loper, M. Marathe, and S. J. Taylor, "Computational Challenges in Modeling & Simulation of Complex Systems," in *Proceedings of the Winter Simulation Conference (WSC '17)*. Las Vegas, NV, USA: IEEE, December 2017, pp. 431–445.

[33] A. Lake. (2014, September) Getting the Most from OpenCL 1.2: How to Increase Performance by Minimizing Buffer Copies on Intel Processor Graphics. [Online]. Available: https://software.intel.com/sites/default/files/managed/f1/25/opencl-zero-copy-in-opencl-1-2.pdf

[34] C. Cummins, P. Petoumenos, M. Steuwer, and H. Leather, "Autotuning OpenCL Workgroup Size for Stencil Patterns," *arXiv preprint arXiv:1511.02490*, November 2015.

[35] X. Liu and P. Andelfinger, "Time Warp on the GPU: Design and Assessment," in *Proceedings of the Conference on Principles of Advanced Discrete Simulation (PADS '17)*. Singapore: ACM, May 2017, pp. 109–120.

[36] N. Baudis, F. Jacob, and P. Andelfinger, "Performance Evaluation of Priority Queues for Fine-Grained Parallel Tasks on GPUs," in *Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '17)*. Banff, Canada: IEEE, September 2017, pp. 1–11.

[37] M. Yang, P. Andelfinger, W. Cai, and A. Knoll, "Evaluation of Conflict Resolution Methods for Agent-Based Simulations on the GPU," in *Proceedings of the Conference on Principles of Advanced Discrete Simulation (PADS '18)*. Rome, Italy: ACM, May 2018, pp. 129–132.