Pedal to the Bare Metal: Road Traffic Simulation on FPGAs Using High-Level Synthesis

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

Görkem Kılınç
gorkem.kilinc@tum-create.edu.sg
TUMCREATE
Singapore

Philipp Andelfinger
philipp.andelfinger@gmail.com
TUMCREATE and Nanyang Technological University
Singapore

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

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

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

ABSTRACT

The performance of Agent-based Traffic Simulations (ABTS) has been shown to benefit tremendously from offloading to accelerators such as GPUs. In the search for the most suitable hardware platform, reconfigurable hardware is a natural choice. Some recent work considered ABTS on Field-Programmable Gate Arrays (FPGAs), yet only implemented simplified cellular automaton-based models. The recent introduction of support for high-level synthesis from C, C++, and OpenCL in FPGA toolchains allows FPGA designs to be expressed in a form familiar to software developers. However, the performance achievable with this approach in a simulation context is not well-understood. In this work, to the best of our knowledge, we present the first FPGA-accelerated ABTS based on widely-accepted microscopic traffic simulation models, and the first to be generated from high-level code. The achieved speedup of up to 24.3 over a sequential CPU-based execution indicates that recent FPGA toolchains allow simulationists to unlock the performance benefits of reconfigurable hardware without the need to express the simulation models in low-level hardware description languages.

CCS CONCEPTS

• **Computing methodologies** → **Agent / discrete models**; **Massively parallel and high-performance simulations**; **Simulation tools**; • **Hardware** → **Hardware accelerators**.

KEYWORDS

FPGA; Traffic simulation; Agent-based simulation; Performance; OpenCL; HLS

SIGSIM-PADS '20, June 15–17, 2020, Miami, FL, USA

ACM ISBN 978-1-4503-7592-4/20/06…$15.00
https://doi.org/10.1145/3384441.3395979

ACM Reference Format:
Jiajian Xiao, Görkem Kılınç, Philipp Andelfinger, David Eckhoff, Wentong Cai, and Alois Knoll. 2020. Pedal to the Bare Metal: Road Traffic Simulation on FPGAs Using High-Level Synthesis. In *Proceedings of the SIGSIM Principles of Advanced Discrete Simulation (SIGSIM-PADS '20), June 15–17, 2020, Miami, FL, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3384441.3395979

1 INTRODUCTION

Agent-based simulation is a modelling approach that simulates actions and interactions of individual autonomous entities called agents. It is widely used to answer what-if questions in a number of areas including chemistry, physics, social engineering, etc. Agent-based Traffic Simulation (ABTS), for example, is a method to investigate traffic flow patterns and to predict the impact of traffic incidents. However, sequential execution of a large-scale ABTS usually incurs substantial run times. Existing research has shown that employing multi-core CPUs or GPUs as accelerators leads to better performance [19]. Efforts have been made to simplify the process of porting sequential code to run on these accelerators [17]. In the last decade, a growing interest can be observed in Field-Programmable Gate Arrays (FPGAs) as high-performance and energy-efficient accelerators for compute-intensive tasks. It has been demonstrated that FPGAs can outperform CPUs or GPUs, e.g., in data encryption [2]. Previous works [12, 14] also show the feasibility of accelerating ABTSs using FPGAs. Those works often rely on converting common traffic models to models based on Cellular Automata (CA), which naturally map to the FPGA's hardware building blocks. In these works, the FPGA applications were expressed in Hardware Description Languages (HDLs), which describe the required functionality at a low level in terms of logical operations and data transferred among hardware registers. The emergence of high-level synthesis toolchains for FPGAs from vendors such as Intel or Xilinx enables the development of FPGA programs in C-like languages, reducing the development effort [11] and enabling portability to and from other hardware platforms.

In this work, we present our design of an FPGA-accelerated ABTS generated from OpenCL code using the SDAccel high-level

synthesis tool by Xilinx. We focus on the FPGA-as-accelerator scenario, i.e., a PC or a cluster equipped with an FPGA accelerator connected via PCI-E. Our code is available online[1].

Our ABTS is based on two key models: a car-following model (CFM) and a lane-changing model (LCM). We consider a proof-of-concept case that still exercises both models: a single road with a configurable number of parallel lanes.

The main contributions of this paper are:

- To the best of our knowledge, we present the first FPGA-accelerated ABTS to rely on models from the traffic engineering literature, and the first to rely on high-level synthesis.
- We describe the main design choices tailored to the FPGA hardware architecture, including an efficient neighbour search mechanism.
- We evaluate the performance using two different high-level synthesis approaches supported by SDAccel.

The remainder of this paper is organised as follows: In Section 2, we describe the fundamentals of FPGAs and discuss related work. In Section 3, we present our approach to accelerate ABTS on FPGAs. We evaluate the performance and the energy consumption in Section 4. Section 5 concludes the paper.

2 BACKGROUND AND RELATED WORK

In this section, we describe the state of art in ABTS and FPGA technologies as well as briefly review the advances in recent literature.

2.1 Agent-based Traffic Simulation (ABTS)

In most of the established ABTSs, the movement of an agent is determined by two models: a car-following model (CFM) and a lane-changing model (LCM). The CFM controls the agent to move forward by following the vehicle ahead and avoiding collisions. In this paper, we use the Intelligent Driver Model (IDM) [13]. IDM receives as input the current velocity, a desired velocity, and the distance and speed of the vehicle in front and returns the acceleration for the next time step.

The LCM triggers lane changes to allow the vehicle to accelerate or to follow its route. In this paper, we employ the MOBIL model, short for Minimizing Overall Braking Induced by Lane change [7]. MOBIL is an incentive-based model that evaluates potential lane-changing manoeuvres and generates a decision that balances the current vehicles acceleration with the required braking by other vehicles.

Both models require gathering the positions and velocities of a maximum of six neighbouring agents: the leaders and followers on the same lane, on the left lane, and on the right lane. In Section 3.1, we propose a simple algorithm that does the neighbour search and agent update efficiently on an FPGA.

2.2 Field-Programmable Gate Array (FPGA)

An FPGA is an integrated circuit consisting of many programmable logic blocks and I/O components. Reconfiguration involves updating the lookup table contained in each logic block to achieve a desired local logical behaviour, and the configuration of the interconnections among the blocks to achieve an overall functionality. FPGAs are used in applications demanding low energy consumption and are more flexible compared to Application-Specific Integrated Circuits [8].

Typically, FPGA programs are expressed in Hardware Description Languages (HDLs). Synthesis tools translate from an HDL to a so-called bitstream that can be transferred to an FPGA. Programming in HDLs requires in-depth knowledge about the hardware and comes with significant design efforts [11]. High Level Synthesis (HLS) tools that generate bitstreams from higher-level code such as C or System C are developed to alleviate this. Programming using HLS in an FPGA-as-accelerator scenario involves two steps: first, an intellectual property describing the main logic of the application needs to be developed; second, a data path that transfers the data from the CPU to the FPGA has to be created. The Open Computing Language (OpenCL) for FPGAs simplifies this process by providing an all-in-one solution. It allows to program FPGAs in a C-like programming language and abstracts away the CPU-FPGA data path by its communication interfaces.

2.3 Open Computing Language and SDAccel

The Open Computing Language (OpenCL) is a parallel programming framework targeting a wide variety of hardware platforms. It allows users to write programs in a C-like language without taking care of hardware details. An OpenCL execution environment typically consists of a host (usually a CPU) and one or multiple devices (e.g., CPUs, GPUs, FPGAs). A host program orchestrates the resources while a device program, consisting of so-called kernels, implements the actual workload. A thread that executes one kernel is called a *work-item*. Work-items are executed in groups named *work-groups*. OpenCL offers a two-layer memory hierarchy: *Local memory* maps to the Block RAM (BRAM) of an FPGA, which is shared among work-items in the same work-group. *Global memory* binds to the massive but latency-prone off-chip memory to which all work-items have access.

Unlike GPUs which utilise Single Instruction Multiple Data (SIMD) parallelisation, an FPGA's parallelisation is two-fold. Each operation (store/load, arithmetic operation, etc.) is compiled to a small circuit called a Functional Unit (FU). An FPGA can generate many FUs for the same operation so that multiple data elements can be processed simultaneously. Further, many FUs are wired together to form a *pipeline*. A work-item steps over one FU at a time while many work-items start successively to fill the different stages of the pipeline (Fig. 1). The start interval between two consecutive work-items is called initiation interval (II). The performance of an FPGA is influenced mainly by two factors: the initiation interval and the operating frequency. Dependencies across work-items may cause a bigger initiation interval. The operating frequency is guided by the cycles spent on the slowest pipeline stage.

The SDAccel tool allows users to express an OpenCL kernel using HDL, HLS (high-level synthesis from C or C++), or OpenCL. In this work, we explore designs with the latter two approaches. The C or C++ used for HLS is close to sequential code in terms of style, with extra syntax provided for FPGA-specific data mappings and optimisation, making portation from existing sequential code trivial. The resulting program presents as a single work-item to the host. The body of an HLS kernel is typically a loop where one data

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

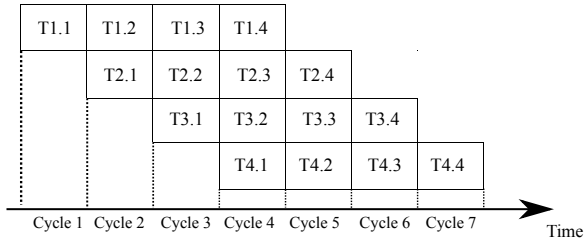

Figure 1: Pipeline Parallelism. We assume four tasks, each task being divided into four FUs (e.g., T1.1 - T1.4).

point is dealt with per iteration. When parallelised on an FPGA, many iterations start a number of cycles apart (as defined by the II), yielding pipelined parallelism. Two types of OpenCL kernels are supported when programming in OpenCL [16]: *Single-work-item* kernels are intrinsically equivalent to HLS with extra use of OpenCL primitives such as *__kernel*, *__global*. *NDRange* kernels are a type of kernel widely used for programming GPUs, and thus offer good portability to and from GPU code. When executed on FPGAs, *NDRange* kernels can either yield SIMD parallelism as GPUs or pipelined parallelism if dependencies exist across work-items. Due to the similarity between kernels written as *Single-work-item* and in HLS, we will omit *Single-work-item* and focus on HLS and *NDRange* for the remainder of the paper.

2.4 Related Work

Methods for accelerating agent-based simulations using hardware accelerators have been widely explored. A comprehensive study on challenges and state-of-the-art solutions on hardware accelerators can be found in [19]. The acceleration of ABTSs using GPUs has been well-studied and substantial speedups have been demonstrated [10, 18]. However, challenges still remain when speeding up ABTSs using FPGAs due to the manifold architectural differences between an FPGA and a GPU, such as smaller memory bandwidths and lower frequencies.

Cong et al. [3] compare the performance of running Rodinia, a widely used GPU benchmark suite, on a GPU and on an FPGA, respectively. They conclude that modern FPGAs can achieve comparable or even better performance for certain tasks. A maximum speedup of 7x over GPU is achieved running a bioinformatics application.

A general FPGA-based discrete event simulation accelerator is presented by Rahman et al. [9]. Their work focuses on the efficient maintaining of the event queues on FPGAs. However, event queues are not necessarily needed in our case.

In previous works, FPGAs are mostly applied to CA-based models owing to the natural mapping of cells to logic blocks [19]. FPGAs have demonstrated performance benefits when applied to CA-based models such as an environment simulation [15], Game of Life [4] and a crowd evacuation model [6]. Schäck et al. [12] implement a traffic simulation on a single lane road based on so-called Global CA model in which a set of cells change their states simultaneously. Tripp et al. [14] develop a CA-based traffic simulation running on a grid of roads. The movement of agents on individual lanes is computed on an FPGA while the agents' transitions from one road to another and at intersections are computed on the CPU. Our implementation does not rely on transforming models to a CA-based format, and thus can be extended to traffic models beyond those discussed in this paper.

3 DESIGN AND IMPLEMENTATION

In a typical ABTS, an agent autonomously performs neighbour searches, makes decisions based on its neighbours' states and updates its own state in each time step. Neighbour search is a crucial step and potentially consumes a significant amount of execution time [18]. Therefore, we start this section with a simple neighbour search algorithm that can be executed efficiently on FPGAs. Then, we discuss two design considerations that can further improve the performance. Lastly, we assemble everything and present our overall design.

3.1 Agent Update and Neighbour Search

As our simulation scenario, we consider a long road with multiple parallel lanes. Initially, agents are assumed to be stored in a per-lane array, sorted by ascending agent positions in driving direction. After a single step of the CFM or LCM is executed, the agents remain sorted: 1) The CFM drives the agent to follow but not to overtake the agent in the front; 2) To carry out a lane change, the lane-changing agent is removed from its original lane and inserted between its leader and follower on the target lane. We present a simple algorithm that relies on the per-lane agent order to efficiently determine the neighbours of all agents on the road. The basic idea is to iterate through the agents in the order of their position considering all lanes, updating per-lane pointers to efficiently identify their neighbours.

Figure 2 illustrates the neighbour search: For each lane l, a pointer $current[l]$ is maintained, initially pointing to the rearmost agent. The neighbour search iterates through the agents by their position across all lanes, until the foremost agent has been reached. Once an agent's neighbours have been identified on lane l, the pointer $current[l]$ is updated to point to the agent's leader on lane l (cf. Figure 3). During the iterations, the invariant holds that the neighbours of each current agent are identified by the *current* pointers as follows: the leader and the follower, if any, on the current lane l are located by $current[l]$+1 and $current[l]$-1 . The leaders and followers on neighbouring lanes l_i, if any, are identified by $current[l_i]$ and $current[l_i]$-1.

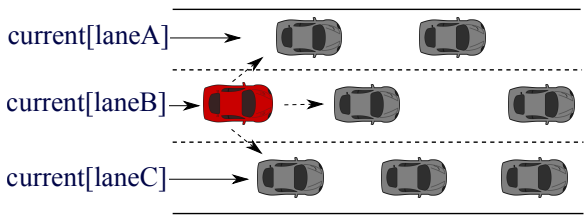

Figure 2: Neighbour search of the first agent (coloured in red). Neighbours are indicated by the dash arrowed lines.

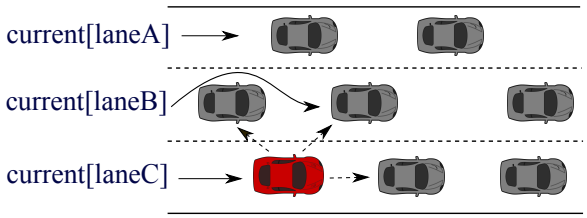

Figure 3: Neighbour search of the second agent (coloured in red). Neighbours are indicated by the dash arrowed lines.

3.2 Design Considerations

To maximise the parallelisation opportunities and tailor the computing workload to the underlying FPGA hardware, we consider the following two design concerns:

3.2.1 Single and Double Buffering. Single and double buffering (SB and DB) are two design principles for ABSs, both of which have implications on parallelized execution, model behavior and the potential for state conflicts [20]. With SB, each agent overwrites its state immediately after the agent behaviours have been executed, whereas with DB, the new agent state is first written to an intermediate buffer and consequently applied for all agents at once. The SB design may cause write-read or write-write dependencies, resulting in larger initiation interval when pipelined on FPGA. Further, with SB, inconsistent agent states may be read as the agent updates gradually progress throughout the pipeline. To avoid these complications, we rely on DB in our implementation.

3.2.2 Number Representation. As the high dynamic range of floating point numbers is not required to represent positions and velocities on individual roads, we rely on fixed-point arithmetic to achieve better power and space efficiency [5]. We use 24 bits to represent the integer part and 8 bits for the fractional part, which enables a minimum number increment of 2^{-8}, i.e., a precision of 0.4cm or cm/s.

3.3 Main Simulation Loop

In the HLS implementation, the main body of the kernel function is a nested loop. The outer loop iterates through the simulation steps. The inner loop, which is pipelined, iterates through the agents to carry out the neighbour search and agent update.

In the NDRange kernel implementation, the loop that counts the simulation steps resides in the host code. The loop that updates the agents is replaced by an NDRange function call. Due to the incremental updating of the *current* pointers in the neighbour search algorithm, SIMD processing of multiple agents is infeasible. However, pipeline parallelization across work-items is possible.

Unintended vehicle collisions may occur due to the employment of DB design [1]: as illustrated in Fig. 5, the red vehicle and the yellow vehicle are close in longitudinal positions and both intend to change lanes to the middle lane. As the vehicles are two lanes apart, they are unaware of each other. As a result, both of them may decide to change to the middle lane, causing a collision. To resolve such conflicts, before writing to the buffer holding the new state, we check whether the last stored agent on the target lane collides with the current agent. The pipelined execution on the FPGA ensures

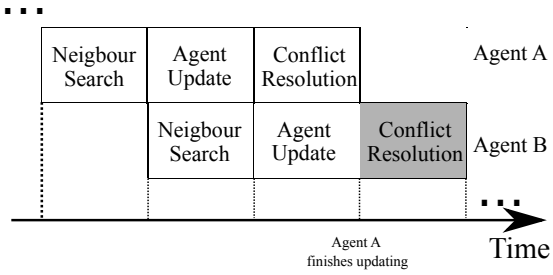

Figure 4: Illustration of pipelined processing of agents

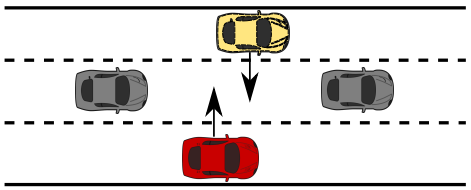

Figure 5: Illustration of a possible collision due to our double-buffering design.

that at this point, the last agent has finished updating its state (cf. Figure 4).

4 EXPERIMENT

We evaluate our HLS and NDRange kernels on an Amazon f1.2xlarge instance equipped with an 8-core Intel Xeon E5-2686, 122GB RAM and an Xilinx Virtex UltraScale+ XCVU13P FPGA. The FPGA has a maximum frequency of 500MHz, 3,780K Logic Cells, 12,288 DSPs and 455Mb RAM. The version of the Xilinx OpenCL Compiler is 2018.2 with GCC version 4.8. Initially, agents are spawned at the leftmost two lanes of the road. We run the traffic simulation on a road with four lanes for 1,000 time steps, varying the number of agents. The run time is compared to that of a sequential CPU implementation on an Intel Xeon-E5 CPU and a SIMD implementation on an NVIDIA GTX 1060 GPU. The proposed neighbour search and conflict resolution implementations rely on the pipelined execution and strict update order guaranteed on the FPGA, however, as GPUs process many agents in an undefined order, we base the GPU implementation on a different fast ABTS design described in [18]. When a conflict is detected, the agent that is further behind on the lane is rolled back to its original lane. As shown in Fig. 6, FPGA-HLS is slower than the CPU at small scales due to the initialisation and data transfer overheads. However, FPGA-HLS outperforms the CPU for 256 and more agents. Interestingly, the GPU performs similarly to the CPU and is slower than FPGA-HLS. This is due to the substantial overheads in the conflict resolution step on the GPU, which constitutes 60-90% of the total execution time. FPGA-HLS achieves a 24.35x speed up over the CPU and 8.9x over the GPU when simulating 16,384 agents. Notably, FPGA-NDRange is the slowest implementation in all cases. One reason for this is the dependencies among work-item on the `current` pointers, which prevent parallel processing of the agents. Moreover, FPGA-NDRange cannot rely on pre-fetching: in the HLS implementation, agent data is pre-fetched from the latency-prone off-chip memory

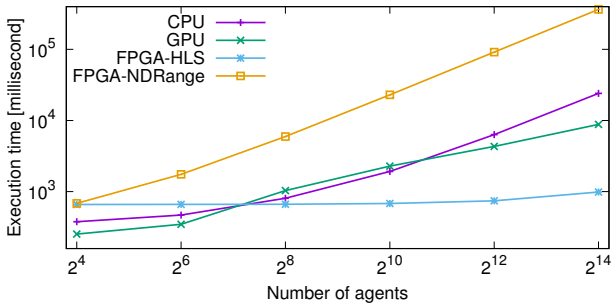

Figure 6: Performance comparison between CPU, GPU, FPGA-HLS and FPGA-NDRange

to the fast BRAM before the main simulation loop is entered. The pre-fetching contributes to the initialisation overhead, resulting in a slowdown at small scales, but a performance increase at larger scales. However, in the NDRange implementation, where multiple work-groups are presented and the BRAM address space can only be shared within the same work-group, the memory required to pre-fetch all agent data for each work-group exceeds the available BRAM at large scales. Pre-fetching a chunk of data per work-group is infeasible as it is only beneficial when the neighbouring agents are pre-fetched, which naturally cannot be done before the neighbour search. Instead, each work-item fetches agent data from the slow off-chip memory, causing a decrease in performance.

Our kernels consume less than 6% of the FPGA's resources. This indicates a good potential to extend our approach to full road network topologies by generating multiple pipelines on the same FPGA, each pipeline dealing with one road. Extra logic might be required in this case for transmitting agents between pipelines.

5 CONCLUSION AND OUTLOOK

In this work, we presented our approach to accelerate ABTS on FPGAs using the SDAccel tool. A neighbour search and a conflict resolution mechanism for FPGAs are proposed. We explored two types of kernels called HLS (Single-work-item) and NDRange as well as two design considerations which help achieve high performance for both kernel types. Our experiments show that FPGA-HLS outperforms a sequential CPU-based simulation due to the pipelined parallelism, and the GPU owing to a smaller overhead in the conflict resolution step. The NDRange kernel achieved low performance, as our neighbour search algorithm is not amenable to automatic parallelization and data pre-fetching in a SIMD fashion. As a result, we recommend the use of Single-work-item or HLS kernels and pre-fetching for workloads similar to our ABTS. In future work, we will extend our approach to be applicable to full road network topologies, which will require a reconsideration of the neighbourhood search algorithm, as well as sorting when agents simultaneously advance to a new road and change lanes.

6 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] Philipp Andelfinger, Jordan Ivanchev, David Eckhoff, Wentong Cai, and Alois Knoll. 2019. From Effects to Causes: Reversible Simulation and Reverse Exploration of Microscopic Traffic Models. In *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS)*. ACM, Chicago, IL, USA, 173–184.

[2] Shuai Che, Jie Li, Jeremy W. Sheaffer, Kevin Skadron, and John Lach. 2008. Accelerating Compute-Intensive Applications With GPUs and FPGAs. In *Proceedings of the Symposium on Application Specific Processors (SASP '08)*. IEEE, Anaheim, CA, USA, 101–107.

[3] Jason Cong, Zhenman Fang, Michael Lo, Hanrui Wang, Jingxian Xu, and Shaochong Zhang. 2018. Understanding performance differences of FPGAs and GPUs. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, Boulder, CO, USA, 93–96.

[4] Lintao Cui, Jing Chen, Yu Hu, Jinjun Xiong, Zhe Feng, and Lei He. 2011. Acceleration of Multi-Agent Simulation on FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '11)*. IEEE, Chania, Greece, 470–473.

[5] Yao Fu, Ephrem Wu, Ashish Sirasao, Sedny Attia, Kamran Khan, and Ralph Wittig. 2016. Deep learning with int8 optimization on xilinx devices. *White Paper* (2016).

[6] Ioakeim G. Georgoudas, Panagiotis Kyriakos, G. Ch. Sirakoulis, and I. Th. Andreadis. 2010. An FPGA Implemented Cellular Automaton Crowd Evacuation Model Inspired by the Electrostatic-Induced Potential Fields. *Elsevier Journal of Microprocessors and Microsystems* 34, 7 (November 2010), 285–300.

[7] Arne Kesting, Martin Treiber, and Dirk Helbing. 2007. General lane-changing model MOBIL for car-following models. *Transportation Research Record* 1999, 1 (2007), 86–94.

[8] Ian Kuon and Jonathan Rose. 2007. Measuring the gap between FPGAs and ASICs. *IEEE Transactions on computer-aided design of integrated circuits and systems* 26, 2 (2007), 203–215.

[9] Shafiur Rahman, Nael Abu-Ghazaleh, and Walid Najjar. 2017. PDES-A: A parallel discrete event simulation accelerator for FPGAs. In *Proceedings of the ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS)*. ACM, Singapore, 133–144.

[10] Daniel Rajf and Tomas Potuzak. 2019. Comparison of Road Traffic Simulation Speed on CPU and GPU. In *23rd IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, Cosenza, Italy, 1–8.

[11] Kyle Rupnow, Yun Liang, Yinan Li, and Deming Chen. 2011. A study of high-level synthesis: Promises and challenges. In *2011 9th IEEE International Conference on ASIC*. IEEE, Xiamen, China, 1102–1105.

[12] Christian Schäck, Rolf Hoffmann, and Wolfgang Heenes. 2010. Efficient traffic simulation using the GCA model. In *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*. IEEE, Atlanta, GA, USA, 1–7.

[13] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. 2000. Congested Traffic States in Empirical Observations and Microscopic Simulations. *Physical review E* 62, 2 (August 2000), 1805.

[14] Justin L. Tripp, Henning S. Mortveit, Anders A. Hansson, and Maya Gokhale. 2005. Metropolitan Road Traffic Simulation on FPGAs. In *Proceedings of the Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '05)*. IEEE, Napa, CA, USA, 117–126.

[15] Ioannis Vourkas and Georgios Ch. Sirakoulis. 2012. FPGA Based Cellular Automata for Environmental Modeling. In *Proceedings of the International Conference on Electronics, Circuits and Systems (ICECS '12)*. IEEE, Seville, Spain, 93–96.

[16] Hasitha Muthumala Waidyasooriya, Masanori Hariyama, and Kunio Uchiyama. 2018. *Design of FPGA-based computing systems with OpenCL*. Springer.

[17] Jiajian Xiao, Philipp Andelfinger, Wentong Cai, Paul Richmond, Alois Knoll, and David Eckhoff. 2019. Advancing Automatic Code Generation for Agent-Based Simulations on Heterogeneous Hardware. In *Proceedings of the European Conference on Parallel Processing*. Springer, Göttingen, Germany.

[18] Jiajian Xiao, Philipp Andelfinger, David Eckhoff, Wentong Cai, and Alois Knoll. 2018. Exploring Execution Schemes for Agent-Based Traffic Simulation on Heterogeneous Hardware. In *Proceedings of the International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, Madrid, Spain, 1–10.

[19] Jiajian Xiao, Philipp Andelfinger, David Eckhoff, Wentong Cai, and Alois Knoll. 2019. A Survey on Agent-based Simulation Using Hardware Accelerators. *ACM Computing Surveys (CSUR)* 51, 6 (2019), 131.

[20] Mingyu Yang, Philipp Andelfinger, Wentong Cai, and Alois Knoll. 2018. Evaluation of Conflict Resolution Methods for Agent-Based Simulations on the GPU. In *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, 129–132.