CELLULAR AUTOMATA BASED TRAFFIC SIMULATION ACCELERATED ON GPU

Pavol Korcek, Lukas Sekanina and Otto Fucik

Brno University of Technology – Faculty of Information Technology
Department of Computer Systems
Bozetechova 1/2, 612 66 Brno
Czech Republic
{ikorcek, sekanina, fucik}@fit.vutbr.cz

Abstract: *Intelligent transportation systems become more and more important with the increasing traffic densities and safety requirements. A reasonably good traffic prediction can be obtained using microscopic traffic simulation models witch distinguish and trace every traffic entity. However, microscopic simulation requires considerable computing resources. In this paper, we propose to accelerate a cellular automata based microscopic traffic simulator using graphic processing units (GPU). The proposed accelerator provides speed-up of 204.65 with respect to a single core solution for problems instances containing 170 mil. cells, equivalent to 935 000 km of traffic network. This solution is sufficient to predict traffic simulations multiple in real-time.*

Keywords: *Traffic, Microsimulation, Cellular automata, Acceleration, GPU, CUDA.*

1 Introduction

Microscopic traffic simulation models, in contrast to macroscopic models, distinguish and trace every single vehicle and driver. The models are very precise and typically employ characteristics such as vehicle lengths, speeds, accelerations, time and space headways, vehicle and engine capabilities, as well as some rudimentary human characteristics that describe the driving behaviour [1], [2].

In the past, the use of microscopic models has been limited in scope and scale of the problem. The applications were restricted to small networks with a limited number of vehicles. With the advent of fast computers a large number of microscopic simulation models have been developed. For an illustration, the SMARTEST report (1999) identified 58 traffic simulation models of which 32 were analyzed [3]. Some of them are true microscopic simulators in the sense that they model the behaviour of each individual vehicle in the traffic flow. Another latest short overview of the microscopic traffic models (2004) can be found, for example, in [4].

Simulation results could be utilized in various so-called intelligent transportation systems (ITS) [7]. Such an example can be driver assistance or collision avoidance systems where simulations are utilized for the traffic state prediction. The most important requirement for these simulators is to simulate the traffic faster than real time, or even better, multiple in real-time. This isn't easily met when traffic is simulated on microscopic level of detail because there exist millions of simulated entities with complex relations among themselves [1], [2], [3], [4], [6]. Moreover, our approach attempts to extend the simulation beyond a few roads and intersections to large areas (e.g. we are going to model and simulate very huge networks such as the whole Czech Republic with approximately 55 000 km of road segments [9]. This length of road segments does not distinguish the directions, the number of lanes, e.g. the real number of kilometres is much higher when including these options. For instance, distance between Prague and Brno city is approximately 200 km, but because there are two lanes in each direction, total length of road segments is four times greater). Hence the acceleration of microscopic models continues to grow in importance.

The goal of this paper is to present an acceleration of microscopic traffic simulation using our advanced cellular automata (CA) based model [8]. For whole model acceleration, a big potential of today's common GPUs is used. In contrast to specialized and expensive accelerators, such as field programmable gate array (FPGA) based machines [10], we propose to utilize a very reasonable solution based on commodity hardware. We retrieved only 3 papers dealing with traffic simulation acceleration on common GPUs. The first one [15] is concentrated on traffic simulation for emergent behaviour. The acceleration of crowd behaviour was shown, which could be possibly mapped to a traffic model. Paper [15] differs from our model in one important aspect -- traffic topology (e.g. intersections) is not modeled. Another paper [16] also deals with traffic simulation but on a different level of detail. Authors showed an efficient execution of field-based vehicular mobility models with their hybrid method on GPU. Generally, the traffic simulation models are faster when the higher abstract level is used [17]. On the other hand, these models are not as precise as the true microsimulators [1], [3] and their approach is therefore not useful in our prediction-based traffic simulations [7]. The last and most recent paper [14] deals with a multi-agent traffic simulation. Different queue-based implementations were analyzed and speed-up of 67 was achieved when compared to a highly optimized Java version. Recall that we do not simulate queues and agents in these queues in our work; we are performing real traffic microsimulations. It means that

we have traffic lanes instead of queues with exact "agent" position on it. Another big difference is that we do not have „traffic on demand", which means to have pre-counted all traffic routes of each vehicle or driver.
The outline of this paper is as follows: After presenting the idea of general purpose computation on GPU in Section 2, the original and advanced model of traffic microsimulation using CA is described in Section 3. In Section 4 a mapping of model to the GPU is presented. Results of experimental evaluation are summarized in Section 5. Finally, conclusions are given in the last section of the paper.

2 General Purpose Computation on GPU

Over the last decade, the graphic cards found in common home PCs have evolved from mere display devices over 3D rendering devices to today's generally programmable devices. Relevant graphics device companies (NVIDIA and ATI/AMD) have come up with frameworks (SDKs) to program GPUs for general problems. These SDKs are named *FireStream* [11] and *CUDA* (Compute Unified Device Architecture) [12] and can be used on graphic cards of their SDK vendors. The presence of SDKs has changed premises rather dramatically. It has become feasible to take a given primarily CPU based algorithm and convert it for GPU execution in a straight way. The problem of porting algorithm in the same way to the various vendors' GPUs has been solved. A framework for writing programs that can be executed across different GPUs and also across heterogeneous platforms (consisting of CPUs, GPUs, and other elements) has been introduced (*OpenCL* [13]). Hence every programmer can easily utilize a modern GPU for general computation. In common, whole processing flow on GPUs consists of four successive steps also shown in Fig. 1:

1. *Copy data from the host memory to the GPU memory.*
2. *CPU instructs the process to GPU.*
3. *GPU executes each processing element in parallel.*
4. *Copy the result from GPU memory to the host memory.*

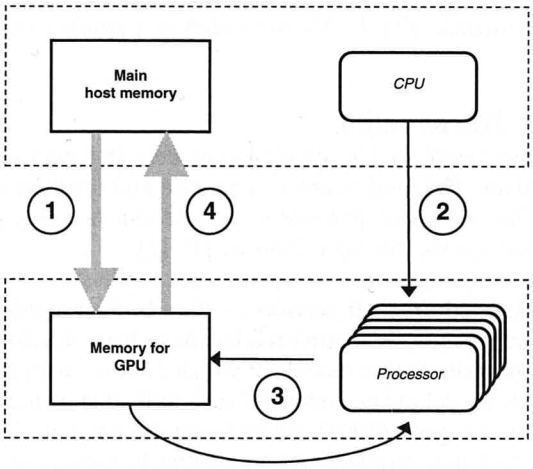

Fig. 1: Processing flow using GPU.

2.1 CUDA

We have chosen CUDA for better compatibility with chosen NVidia graphic card (see Tab.1). This SDK with the new Fermi architecture [31] promises the best achievable speed-ups on GPU and an active community of developers [12]. CUDA can be performed on any NVidia graphic card from the *GeForce-8* generation on both Linux and Windows operating system.

2.1.1 SW model

From software point of view, the CUDA framework is an extension to the standard C language that enables us to write code for CPU and GPU in the same file. It therefore is rather easy to program in for any experienced C/C++ programmer. It basically adds the keywords *__device__*, *__global__*, *__host__* as method decorators to indicate whether the methods are run on the host CPU or the GPU and from where they could be called. Additionally it adds own syntax for describing with how many parallel threads a method should be started. The methods declared as a global are so called "kernels". These kernels could be called from the "host" (the PC/CPU the graphics device is running in) and run on the GPU. These kernels run simultaneously on different data sets in multiple threads of execution because GPU is a massively parallel processor, which supports thousands of active threads [12]. As we only have a limited number of "real" hardware processors, threads are joined to thread blocks, which again are bundled to a grid of thread blocks. Hardware thread scheduler manages recently up to 1536 simultaneously active threads for each streaming multiprocessor across 16 kernels [32]. Threads are sharing a set of registers as well as a rather small on-chip memory area. Threads running in the same block can be synchronized with each other, whilst threads amongst block of threads cannot be easily synchronized. Threads within a block can also access a small amount of additional local shared memory. Though only small amount of threads can make up one block of threads the grid of blocks can run thousands or even trillions of threads in parallel as can be seen in Fig. 2. More details about CUDA application programming can be found in [12].

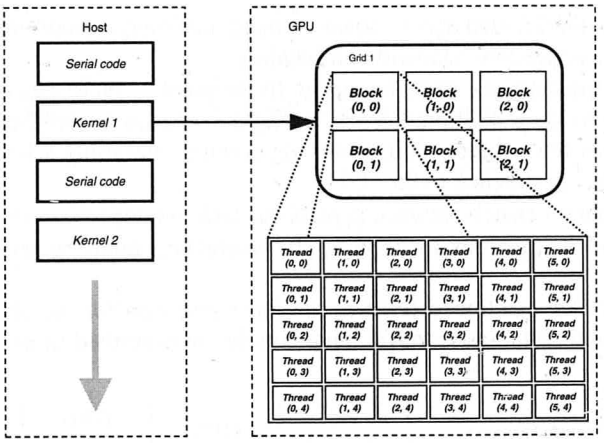

Fig. 2: CUDA programming model.

2.1.2 HW model

Today's graphic cards are connected via PCI Express system bus to the host computer. However, this bus is a potential bottleneck or possible source of big latency in such applications where a huge number of transfers are needed between the GPU and the host CPU [18]. The architecture of the NVidia series GPU is shown in Fig. 3. There is an on-board main memory (with GB capacity). It also acts as an entry or output point during the communication with CPU. Unfortunately, big capacity is outweighed with high latency and this memory is not cached anywhere. The second part is a graphic chip, which consists of N symmetric multiprocessors (SM) each based on SIMD (Single Instruction Multiple Data) architecture. These SMs have their own on-chip shared memory which can be accessed by all the threads of a thread block, texture and constant memory which are "readily" cached on-chip, built-in hardware based scheduler and also shared instruction unit with a hardware thread scheduler. Each SM consists of M functional units (the newest NVidia Fermi GPUs have 32 functional units per single SM [32]), which have own set of registers. These registers are the fastest memory components in the graphic card memory hierarchy model, but on the other hand, they have the smallest capacity.

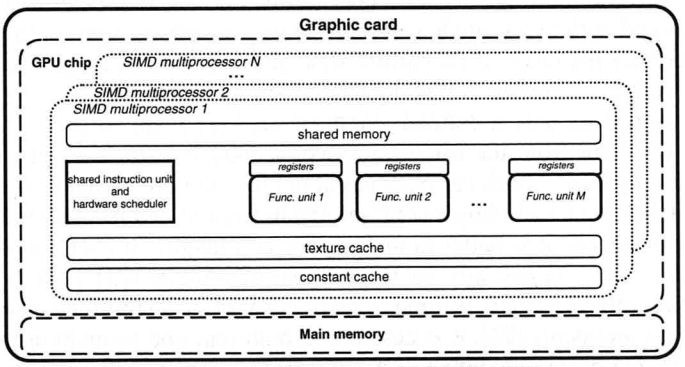

Fig. 3: The architecture of NVidia GPUs.

2.1.3 Improvements in a new generation NVidia Fermi GPUs

The onfiguration of our test system is shown in Tab. 1. As it can be seen, the new Fermi architecture [31] is in place with our *NVidia GTX 480* graphic card. Fermi, compared to previous GPU architectures, offers some improvements:

- *Indirect branching,* which is common for CPUs, but was previously prohibited for NVidia GPUs.
- *Single unified address space* for local (thread private), shared (for each thread block) and global memory (device and system-wide) is used in this generation NVidia hardware.
- *Increased register file* to 32 000 entries or 128 kB in total. The number of registers per execution unit is dropped by half, to 1 000 entries, versus the previous NVidia GPU generation. The register file is designed to sustain full 32-bit throughput every clock cycle.
- *L1 cache per each core* that can be either configured as 16 kB or 48 kB in size. The L1 caches are tied together with a shared L2 cache. Shared memory and L1 data cache (L1D) are respectively used for explicit and implicit communication between threads. The array can be configured with a 16 kB or 48 kB shared memory, with the remainder used for L1D cache.
- *L2 cache* (768 kB) and extra atomic execution units are used to accelerate atomic operations. Fermi's unified L2 cache serializes atomic operations in a different manner, to reduce the number of memory write backs. For

non-atomic operations, the L2 also acts to coalesce many different memory requests (e.g. from a whole warp) into fewer transactions, improving bandwidth utilization.

- *Newer scheduler* can maintain the state for up to 16 different kernels, one per SM. Each SM runs a single kernel, but the ability to keep multiple kernels in flight increases its utilization, especially when one kernel begins to finish and has fewer blocks left. More importantly, assigning a kernel per core means that smaller kernels can be efficiently dispatched to the GPU.
- *Reduced latency* for context switch between kernels has been reduced 10 times (to around 25 microseconds).
- *Full predication* for all instructions to improve the instruction fetch by removing bubbles caused by taken branches.

As it can be seen, all of previously mentioned features are more common for the standard CPU architecture. Traffic simulation, especially cellular automata based traffic microsimulation described in next section, can potentially highly exploit from all of them.

	CPU	PC memory	GPU	GPU memory	OS + SW
System configuration	*Intel Core i7 920@3.3GHz*	*12 GB*	*GTX 480 with 15 SM*	*1.5 GB*	*Ubuntu 10.4 LTS x64 Cuda 3.0, gcc 4.1 (with CUDA support)*

Tab. 1: Test system configuration.

3 Cellular automata based traffic simulation

Cellular automata based models have become popular in the area of microscopic traffic flow simulations because of their simplicity and suitability for acceleration. The first highly suitable model for acceleration of single-lane freeway traffic was introduced by Nagel and Schreckenberg in 1992 [19]. Because of the model's simplicity, it is possible to perform millions of updates in a second [20]. Thus, it may be used for simulating a high volume of traffic over very large networks. Due to some critics of unrealistic behavior (such as [21]), simple CA based models are often extended. On the other hand, CA models were shown to be able to capture all basic phenomena that occur in traffic flows [22], not only in the field of vehicular traffic flow modeling, but also in other fields such as pedestrian behavior, escape and panic dynamics, etc.

Nagel's and Schreckenberg's traffic model [9] was initially defined on a one-dimensional array of cells with open or periodic boundary conditions and with every single cell representing a road segment. A local transition function (a rule) defines the new state of a cell on the basis of its current state and the state of neighboring cells. Globally viewed, it describes the movements of vehicles from one cell to another cell in a discrete way.

In space domain, each cell represents only a defined length of road segment, so space is coarse-grained. This coarse graininess is fundamentally different from the usual microscopic models, which adopt a semi-continuous space. The length of 7.5 m is mostly chosen because each car occupies about this amount of space in a complete jam [9]. So this is the average length of a car on the road, including a constant gap in front of and behind each car. In order to model other kinds of vehicles in CA based models, it is better to use another cell length, or even more precisely, use more smaller cells for representing a single vehicle -- especially for bigger types of vehicles [23], [24].

Each CA simulation step represents an amount of time in reality. Based on suggestions about driver reaction time given in [25], [26] and field data measurements [23], it is possible to represent one simulation step between 0.6 and 1.5 s. In the CA model this value is crucial because together with cell length it gives us the granularity of minimal model speed jumps[1].

Normally, urban traffic road networks are very complex. Paper [27] has shown that arbitrary kinds of road and intersections can be reduced to only a few basic elements. For constructing more complicated traffic networks, we shall simply connect more various kinds of cells to the desired topology. So it is possible to build more traffic lanes [28], roundabounds [29] or very complex topologies for whole cities [30]. Fig.4. shows an example of complicated road intersection modeled using CA.

Properties of single lane traffic are originally modeled on the basis of integer valued probabilistic cellular automaton rules [20]. The local transition function can be formalized as follows: each vehicle in a cell has an integer velocity v_v with values between zero and v_{max}. Real vehicle speed v_{act} is multiplication of minimal speed jump and actual vehicle velocity v_v.

We let $\gamma(i)$ denote the cell gap in front of cell i. For an arbitrary configuration, one update of the simulation system consists of the following four consecutive steps, which are performed in parallel for all vehicles [19]:

1. **Acceleration:** *if* $(v_v < v_{max})$ *and if* $(\gamma(i) > v_v)$ *then* $v_v = v_v + 1$.
2. **Slowing down:** *if a vehicle at site i sees the next vehicle at site* $i + j$ $(j \leq v_v)$, *it reduces its speed, so* $v_v = j - 1$.
3. **Randomization:** *with probability p, the velocity of each vehicle (if* $v_v > 0$*) then* $v_v = v_v - 1$.
4. **Car motion:** *each vehicle is advanced* v_v *sites.*

[1] For instance, if simulation step is *1 s* and cell length is *5 m* [19], minimal speed jumps are 5 m/s (27 km/h).

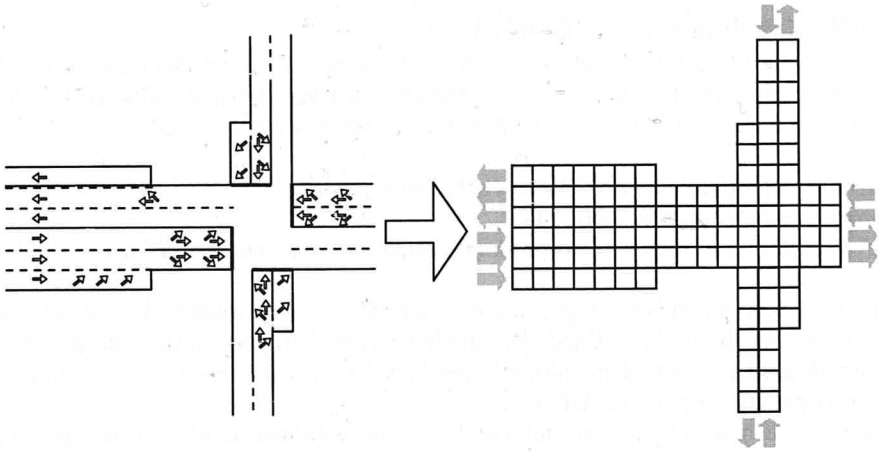

Fig. 4: A road intersection (left side) modeled with the celluiar automata (right side).

This simple CA based traffic model shows nontrivial and realistic behavior of car-following principle [19]. Step 3 is essential in simulating traffic flows since the dynamics is completely deterministic, and without this randomness, every initial configuration of vehicles and corresponding velocities reaches a stationary pattern which is shifted backwards. This simple model is capable of reproducing characteristic properties of real traffic, such as certain aspects of flow-density relation, spatio-temporal evolution of jams, stop-and-go waves. More details can be found in [19], [6].
For multilane traffic simulation not only car-following rules have to be applicable. The behavior of car overtaking should be considered as an additional set of rules to the basic model. Such an example can be found in paper [28] where each vehicle is able to overtake other vehicle only when its speed is sufficient and there is enough space for safe overtaking. These basic rules follow the field behavior of drivers.

3.1 Advanced cellular automata model

Basic cellular automata traffic microsimulation model has been updated to the advanced cellular automata in previous work [8], in order to *a) adapt simulation model to local conditions, b) achieve better precision of model* and finally *c) accomplish faster microsimulations.* Based on local measurements, cell length has been reduced from 7.5 m to 5.5 m. The simulation time has been adjusted from 1 sec. to 1.2 sec according to average driver reaction time [23]. Both these values give minimal speed jump of 16.5 km/h. Cell neighbor is 11, and together with cell length it gives a maximal distance of 60.5 m between the leader and the follower. As observed, for example in [25], for separations greater than 61 m, car-following is negligible on a driver's behavior. For cell state encoding a bit level encoding utilizing only one 16-bit integer has been used. Finally, double-buffering technique for previous and next cell state storage has been utilized.

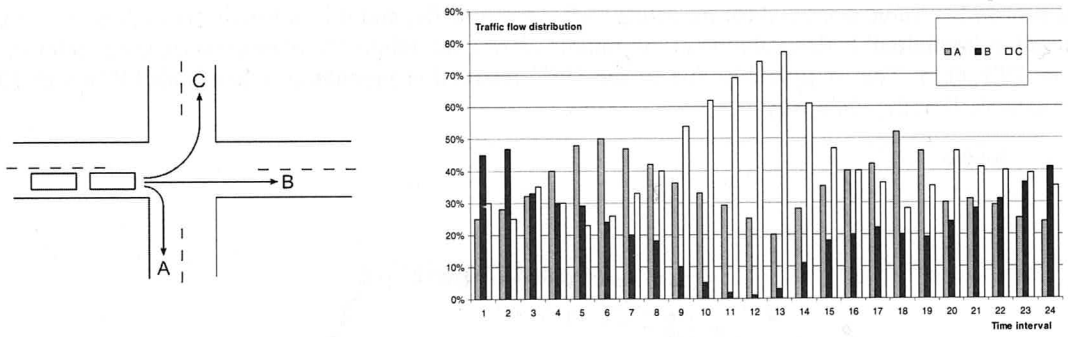

Fig. 5: Simple road intersection (right side) with traffic flow distribution over the time (left side).

From the global point of view the behavior on road intersections should be defined. In the original model, this was done using traffic routes for each vehicle [19]. There isn't available such information in our case, hence another solution must be considered. Based on data provided from our industrial partner we are able to measure an average traffic flows on selected road intersections. An example of flow distribution on a simple road intersection over time is shown in Fig. 5. Each incoming vehicle (Fig. 5 on the left) has one of three (A, B or C) chance to turn and the probability (on the y-axis) of turning vehicle in any direction is various in different time interval (on the x-axis).
All other details about advanced cellular automata can be found in paper [8]. This updated model has been already proposed to model road traffic more realistically and also quite efficiently employing standard CPU based multicore devices, where, depending on the number of processing elements, a nearly linear speed-up has been achieved [8].

4 CA based traffic simulation using GPGPU

Traffic microsimulation with advanced cellular automata model needs to run an identical local transition function on different cells (that can be considered as different data), meaning that this is exactly what GPUs have been designed to speed-up. Proposed microsimulation was implemented in three different ways using GPU:

a) *road intersection cells simulated on CPU and other cells in GPU,*
b) *traffic network simulated only in GPU,* and
c) *scenario of b) with standard operations (adding, subtraction and comparison) instead of bit level operations.*

The implementation (a) exploits a relatively huge amount of data transfers between GPU memory and the host memory, but the synchronization can explicitly be defined. The implementation (b) gets all the data to the GPU main memory. Simulation ends when desired number of iterations is reached. The implementation (c) is proposed to compare the effectiveness of various operator types on the GPU.
Each cell state is stored in 8 bytes (16 bits is multiplied by 2 for information redundancy used in advanced cellular automata model for better accuracy [8] then multiplied again by 2 to employing the double-buffering technique). In both cases, a simple traffic network model has been implemented, where a road intersection occurs every 500 m. This value reflects the mean length of roads after which road intersection occurs in reality [9]. Every CA cell knows if it is the input cell to the road intersection based on its position. Supposing 24 time intervals on every road intersection (see Fig.5 above) and supposing 16 bits used for encoding one probability value in one direction (of 3: A, B or C), we need 768 bits (96 B) for encoding one road intersection which are stored separately for every intersection on host memory and cached (a) or in GPU main memory and also cached (b, c).

5 Results

To measure the speed-up of the proposed solution each of implemented versions was tested on our system. We recorded the start time and after reaching demanded number of simulation steps (equal to 42 min), the execution of program was terminated and results were sent to CPU. After this, the running time was recorded (including time of bus transfers). Fig. 6 shows the achieved speed-up compared to a single core version running on *Intel Xeon CPU5420 @ 2.5GHz* [8]. The speed-up of 204.65 was achieved in the fastest implementation. The fastest solution is the microsimulation implemented entirely on the GPU utilizing standard operations. The slowest implementation is running on both CPU and GPU. One very important finding is that the GPU solution is faster (relatively to the CPU version) when simulating higher traffic density. Another finding is that the GPU solution is ineffective in case when the size of problem (number of CA cells) is small. It is probably because the initialization phase is very time consuming. Consequently, the simulated time should be at least 42 min (for the smallest problem size -- 250 000 cells). This is because of the initialization phase overhead.
Other experimental results can be seen in Fig. 7. We used the fastest implementation with the best speed-up (simulating 170 mil. of CA cells) to evaluate dependency of the GPU acceleration on the *p-value* change. As it was explained in Section 3, this probability value is crucial for modeling traffic realistically, and it is normally set to 30%. We found out, that the acceleration is minimal if this value is in the middle (50%) and relatively symmetric on other sides (e.g. 30% almost same as 70% etc.). This is probably due to the GPU instruction prediction, when both (30% and 70%) are predicted (and executed) in the same way.

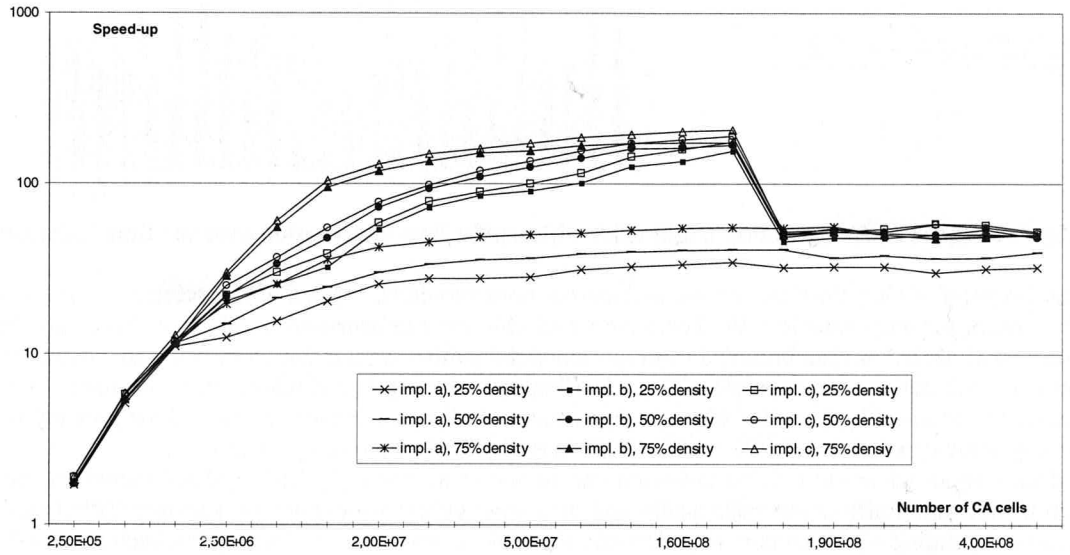

Fig. 6: GPU speed-up of three different implementations with respect to a single core version running on the Intel Xeon E5420 @ 2.50 GHz. Each run is 42 minutes of the real traffic.

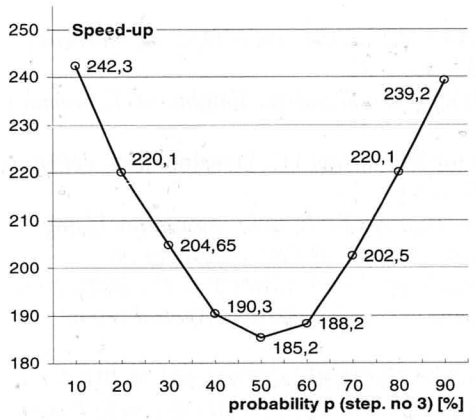

Fig. 7: GPU speed-up with different algorithm settings. P-value on the y-axis is the probability of randomization step in the CA microscopic traffic simulation model.

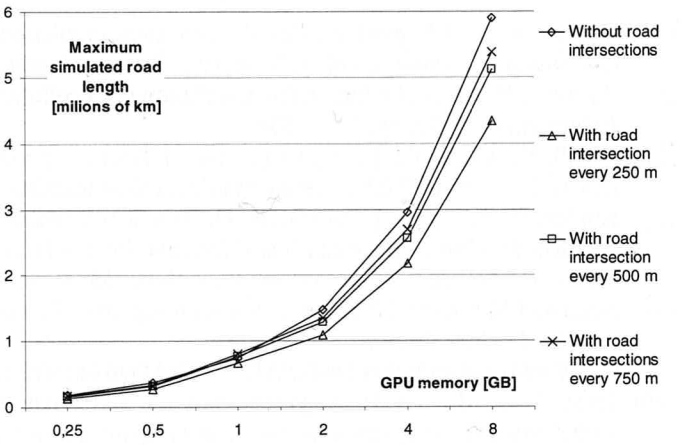

Fig. 8: The maximum theoretical length of simulated road network depending on the GPU memory size.

Fig. 8 shows the maximum theoretical length of simulated road network (with or without intersections) depending on capacity of the GPU memory. This maximum was nearly achieved ("nearly" because the GPU memory is also used for storing another kind of data -- programs, etc. [33]).

The estimated efficiency of simple multicore version of cellular automata based traffic microsimulation introduced in [8] is shown in Fig. 9. For comparison, efficiency of solution presented in this paper is also shown. The number of processing elements in our GPU is used (480) as the number of cores. A simple CPU multicore version is more effective than our GPU implementation, but on the other hand, real results were obtained unig 4 and 8 cores; the remaining values were estimated based on proposed efficiency.

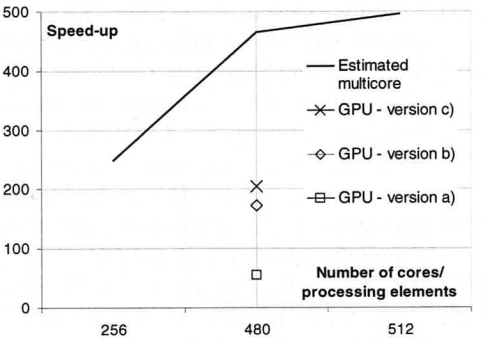

Fig. 9: Efficiency of the simple multicore version [8] and the GPU versions. All of them are with the same configuration of the model.

6 Conclusion

In this paper, it was shown that modern GPU can be effectively used for acceleration of microscopic and cellular automata based traffic simulation. The acceleration profits from the new Fermi architecture. On the other hand, there exist also some limitations. It was shown that the bus traffic is a bottleneck and therefore all data should be stored at least in the relatively big GPU memory (1.5 GB). Moreover, this also means that the GPU memory makes limitations to size of simulated road length (see speed-up drop in Fig. 6). Even this limitation, we are able to simulate 17 times greater traffic area than the traffic network of the Czech Republic [9] multiple in real-time. Finally, we can claim that our GPU version is still reasonable for traffic microsimulation even if multicore version has a higher efficiency. It is because there isn't any multicore device with a suitable number of cores.

Acknowledgement: This work was partially supported by the Czech Science Foundation under projects P103/10/1517 and GD102/09/H042, by the research program MSM 0021630528 and the BUT grant FIT-S-11-1.

References:

[1] Maerivoet, S. and De Moore, B., Transportation planning and traffic flow models, *Technical Report No. 05-155,* Katholieke Universiteit Leuven, 2005.

[2] Brackstone, M. and McDonald, M., Car-following: a historical review, *Transportation Research Part F: Traffic Psychology and Behaviour,* vol. 2, no. 4, Elsevier, 1999, pp. 181 – 196.

[3] Algers, S., et. al., Simulation modelling applied to road transport European scheme tests (SMARTEST) – Review of Micro-Simulation Models, *SMARTEST Project Deliverable D3*, Institute for Transport Studies, University of Leeds, March 1998.

[4] Aycin, M., F. and Benekohal, R., F., Linear acceleration car-following model development and validation, *Transportation Research Record: Journal of the Transportation Research Board,* vol. 1644, Washington, 1998, pp. 10 – 19.

[5] Dupuis, A. and Chopard, B., Cellular automata simulations of traffic: A Model for the city of Geneva, *Networks and Spatial Economics,* vol. 3, Number 1, Springer Netherlands , 2003, pp. 9 – 21.
[6] Barlovic, R., et. al., Online traffic simulation with cellular automata, *Traffic and Mobility: Simulation–Economics Environment,* 1999, pp. 117 – 134.
[7] Boxill, S., A. and Yu, L., An Evaluation of Traffic Simulation Models for Supporting ITS Development, *Technical report,* Number 167602-1, Texas Transportation Institute, Texas, 2000. 116 p.
[8] Korček, P., Sekanina, L. and Fučík, O., Towards Scalable and Accurate Microscopic Traffic Simulation Using Advanced Cellular Automata Based Models, *Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems Workshops*, Madeira, IEEE ITSC, 2010, pp. 27-35, ISBN 978-972-8822-20-0.
[9] Road and Motorway Directorate of Czech republic, Roads and motorways in Czech Republic, *Annual report,* Chapter 1, 2009, Online: http://www.rsd.cz/rsd/rsd.nsf/0/AE55C3DAD269424BC12575CB0050A3A7/$file/RSD2009en.pdf [9/2010]
[10] Tripp, J., L., Morveit, H., S., Hansson, A., A. and Gokhale, M., Metropolitan road traffic simulation on FPGAs, *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05),* 2005, pp. 117 – 126.
[11] AMD FireStream www page. AMD Accelerated Parallel Processing (APP) SDK (formerly ATI Stream), Online: http://developer.amd.com/gpu/AMDAPPSDK/Pages/default.aspx [3/2011].
[12] Nvidia CUDA www page. CUDA Zone, Online: http://www.nvidia.com/object/cuda_home_new.html, [3/2011].
[13] Khronos OpenCL www page. OpenCL - The open standard for parallel programming of heterogeneous systems, Online: http://www.khronos.org/opencl/ [2/2011].
[14] Nagel, K. and Strippgen, D., Using common graphics hardware for multi-agent traffic simulation with CUDA, *Proceedings of the 2nd International Conference on Simulation Tools and Techniques (Simutools '09),* ICST 2009, Belgium, 2009, pp. 1 – 8, ISBN: 978-963-9799-45-5.
[15] Passos, E., et. al., Supermassive crowd simulation on GPU based on emergent behavior. *Proceedings of the Seventh Brazilian Symposium on Computer Games and Digital Entertainment,* 2008, pp. 81 – 86.
[16] Perumalla, K., S., Efficient Execution on GPUs of Field-Based Vehicular Mobility Models, *22nd Workshop on Principles of Advanced and Distributed Simulation (PADS '08),* Roma, 2008, pp. 154 – 155.
[17] Immers, L., H., Logghe, S., Traffic Flow Theory, Katholieke Universiteit Leuven, English version, 2005. p. 39.
[18] Ueng, S., Z., Lathara, M., Baghsorkhi, S., S. and Hwu, W., W., *CUDA-Lite: Reducing GPU Programming Complexity, Lecture Notes in Computer Science: Languages and Compilers for Parallel Computing,* Vol. 5335, 2008, pp. 277 – 288.
[19] Nagel, K. and Schreckenberg, M., A cellular automaton model for freeway traffic, *Journal de Physique I*, vol. 2, Issue 12, 1992, pp. 2221 – 2229.
[20] Nagel, K., High-speed microsimulations of traffic flow, Ph.D. thesis, University of Cologne, 1995.
[21] Chakroborty, P., and Maurya, A., K., Microscopic analysis of cellular automata based traffic flow models and an improved model, *Transport Reviews,* vol. 28, issue 6, 2008, pp. 717 – 734.
[22] Chowdhury, D., Santen, L., Schadschneider, A., Statistical physics of vehicular traffic and some related systems, *Physics Reports 329,* 2000, pp. 199 – 329.
[23] Bham, G., H. and Benekohal, R., F., A high fidelity traffic simulation model based on cellular automata and car-following concepts, *Transportation Research Part C: Emerging Technologies,* vol. 12, Elsevier, 2004, pp. 1 – 32.
[24] Hafstein, S. F., et. al., A high-resolution cellular automata traffic simulation model with application in a freeway traffic information system, *Computer–Aided Civil and Infrastructure Engineering,* vol. 19, no. 5, Blackwell Publishing, 2004 , pp. 338 – 350.
[25] Aycin, M., F. and Benekohal, R., F., Comparison of car-following models for simulation, *Transportation Research Record,* vol. 1678, Washington, 1999, pp. 116-127.
[26] Aycin, M. F. and Benekohal, R., F., Linear acceleration car-following model development and validation, *Transportation Research Record: Journal of the Transportation Research Board,* vol. 1644, Washington, 1998, pp. 10 – 19.
[27] Esser, J. and Schreckenberg, M., Microscopic simulation of urban traffic based on cellular automata, *International Journal of Modern Physics C,* vol. 8, Issue 5, 1997, pp. 1025 – 1036.
[28] Nagel, K., Wolf, D., E., Wagner, P. and Simon, P., Two-lane traffic rules for cellular automata: A systematic approach, *Physica E,* vol. 58, Issue 2, 1998, pp. 1425 – 1437.
[29] Campari, E., G., Levi, G. and Maniezzo, V., Cellular automata and roundabout traffic simulation, *Cellular Automata,* ser. LNCS, vol. 3305, Springer Berlin, 2004, pp. 202 – 210.
[30] Dupuis, A. and Chopard, B., Cellular automata simulations of traffic: A Model for the city of Geneva, *Networks and Spatial Economics,* vol. 3, Number 1, Springer Netherlands, 2003, pp. 9 – 21.
[31] NVidia Fermi www page. Next generation CUDA Architecture, Online: http://www.nvidia.com/object/fermi_architecture.html, [4/2011].
[32] Glaskowsky, P., N., NVidia's Fermi: The First Complete GPU Computing Architecture, White Paper, 2009.
[33] Nguyen, H., GPU Gems 3. Addison-Wesley Professional, 2007, ISBN 978-0321515261.