Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 151 (2019) 726–732

Procedia Computer Science

www.elsevier.com/locate/procedia

The 8th International Workshop on Agent-based Mobility, Traffic and Transportation Models, Methodologies and Applications (ABMTRANS)
April 29 - May 2, 2019, Leuven, Belgium

GPU-based Parallel Computing for Activity-based Travel Demand Models

H. Zhou[a,*], J.L. Dorsman[b], M. Snelder[a], E. de Romph[a], M. Mandjes[b]

[a] *Sustainable Urban Mobility and Safety, Dutch Applied-Science Organization (TNO), Den Haag, The Netherlands*
[b] *Korteweg–de Vries Institute for Mathematics, University of Amsterdam, Amsterdam, The Netherlands*

Abstract

Activity-based travel demand models (ABMs) are gaining popularity in the field of traffic modeling because of their high level of detail compared to traditional travel demand models. Due to this, however, ABMs have high computational requirements, making ABMs hard to use for analysis and optimization purposes. We address this problem by relying on the concept of parallel computing using a computer's graphics processing unit (GPU). To illustrate the potential of GPU computing for ABM, we present a pilot study in which we compare the observed computation time of an ABM GPU implementation that we built using NVIDIA's CUDA framework with similar, non-parallel implementations. We conclude that speed-ups up to a factor 50 can be realized, enabling the use of ABMs both for fast analysis of scenarios and for optimization purposes.

Keywords: Parallel computing; GPU; Activity-based travel demand modeling; Speed-up; ActivitySim;

1. Introduction

With the advent of new mobility concepts such as Mobility as a Service (MaaS), travel demand models are indispensable as they forecast travel demands of the population under alternative circumstances. These forecasts allow traffic supply models to assess the future performance of traffic networks. Activity-based travel demand models (ABMs) view travel demand as a result of people's desires to participate in activities. Based on behavioral theories, they predict the activities which individuals will undertake, which then translate to travel demand forecasts. ABMs stand out as compared to traditional travel demand models due to their high level of detail. This is partly due to the fact that ABMs model activity behavior on an individual level, rather than on an aggregated level. They do this by keeping track of

[*] Corresponding author. Tel.: +31-88-86-68165.
E-mail address: han.zhou@tno.nl

10.1016/j.procs.2019.04.097

individual agents separately. More particularly, for this purpose, ABMs may include probabilistic choice models to allow individuals to have mutually different behavior, although they might have the same behavior characteristics. As a result, ABMs tend to make more reliable travel demand forecasts. As mentioned in [1], ABMs and agent-based modeling are intimately related. For more background, see [2, 3, 4].

The high accuracy level of ABMs, though, does come at the price of high computational requirements. An ABM studied in [5] involving a scenario with a population size of three million is reported to take 4 hours to run a single iteration on an Intel Xeon processor with 16 cores, 32 threads at 3.4GHz and 64GB of RAM. Due to the stochastic nature of the model, however, many runs are required to produce reliable forecasts. On top of this, the use of ABMs for optimization purposes requires the analysis of many scenarios. In such cases, it is clear that the computation time of ABMs becomes unacceptably high. In this paper, we address this problem by presenting a method to reduce the computation time of ABMs.

In the literature, several attempts to reduce computation times have already been made [6]. Most aim to do this by reducing the level of detail, for example by running the ABM only with a fraction of the population [7] or by reducing the number of possible activity locations [8]. However, this detail reduction is not desired, as this reduces the accuracy of the forecasts. Therefore, another strategy for computation time reduction is required. For a related model based on genetic algorithms, such a strategy can be found in [9], which is based on the use of a computer's graphics processing unit (GPU). GPUs can also be used to reduce computation times in MATSim, which is an open-source framework implementing large-scale agent-based transport simulations [10].

Inspired by this, we propose in this paper the use of GPU-based parallel computing in ABMs. We contribute to the existing literature by explaining how to implement parallel computing using the fact that an ABM is comprised of a large number of independent computations, which can be done in parallel. This independence stems from the fact that activity schedules of individual persons are mostly uncorrelated, so that these schedules can be generated concurrently. ABMs do acknowledge the fact that dependence may arise between persons within a particular household. In these cases, however, parallel computing can still be incorporated on a household level. We propose the use of a computer's GPU instead of its CPU, as GPUs are designed for parallel computation purposes. To our knowledge, the use of parallel computing using a computer's GPU, has not been considered before in an ABM context in combination with discrete choice models.

Next to explaining how GPU-based parallel computing can be done in ABMs, we explore the speed-up in computation time that can be realized. To this end, we implemented GPU-based parallel computing in one of ABM's computational components using NVIDIA's Compute Unified Device Architecture (CUDA) [11]. By using the CUDA framework, a function can be executed on many data elements in parallel. Hence, by exploiting the independence properties mentioned above, households and/or persons can be processed in parallel by many different CUDA threads. We performed a pilot study, from which we conclude that for realistic scenario sizes, speed-ups of up to 50 in terms of computation time can be realized compared to a similar non-parallel implementation based on the open-source ABM-package ActivitySim [12]. This speed-up relieves the above-sketched problem, and paves the way for the extensive use of ABM in the analysis and optimization of any given model scenario.

The remainder of this paper is organized as follows. Section 2 provides an explanation of how ABMs work. Then, we explain in Section 3 how GPU-based parallel computing can be implemented into the framework of ABM. Subsequently, Section 4 presents the pilot study, after which conclusions are presented in Section 5.

2. The activity-based travel demand model

In this section, we consider ABMs in more detail. The goal of an ABM is to predict a complete activity schedule for every member of the population on a given day, which in turn leads to travel demand forecasts. After a preliminary phase where the scenario along with its population is built using input data (surveys, etc), ABM essentially consists of four components making subsequent choices for each person which fulfill the goal. The first component makes several long-term choices for each person (e.g. concerning ownership of a car, the location of school or work, etc.), based on which the second component chooses for every individual on a daily level what the main activity purpose will be (work, leisure, etc). Given this purpose, the third component subsequently generates and schedules the tours each person that day undertakes (e.g. home-work-home), along with the preferred travel mode (e.g. car, train, etc).

The final component then decides on the individual trips that make up this tour, and schedules the trips during the day and the travel mode of each trip.

While all components make decisions for each person on different levels of detail, they are typically made using the same discrete choice model such as the multinomial logit model. In this model, each alternative i from an alternative set $\mathcal{I}$ is associated with a utility value V_i that is computed by

$$V_i = \alpha_i + \sum_{j=1}^{N} \beta_{i,j} C_j. \tag{1}$$

Next to an alternative-specific constant α_i, the utility V_i is driven by a linear combination of the attributes $C_1, \ldots, C_N$ of the chooser (think of a person's age, size of household, etc.) and the $\beta_{i,j}$ represent their coefficients for alternative i. The determination of the constant and the coefficients in (1) comprises an area of research that is beyond the scope of this paper. It is worth emphasizing, however, that fitting a utility function to reality is never done perfectly, and thus an error term ϵ_i should be included to represent the non-observable utility of an alternative. The multinomial logit model assumes that a traveler will choose the alternative i with the highest utility $U_i = V_i + \epsilon_i$. Furthermore, it is assumed that the error terms are mutually independent and Gumbel distributed. From these assumptions, it follows that the choice for alternative i is made with probability

$$P_i = \frac{e^{V_i}}{\sum_{j \in \mathcal{I}} e^{V_j}}. \tag{2}$$

Through simulation, the choices in each of the components can now be sampled. This leads to scheduled activities for each person in the population, which translates directly to travel demand forecasts. Because of the stochastic nature of this approach, multiple model iterations are required to obtain reliable forecasts. For more background on the components of ABM and its underlying discrete choice models, see Chapter 3 in [3].

For illustrative purposes, we now focus on the second mentioned component, the daily main activity purpose (DAP) component, in more detail. This is the component in which we will implement GPU-based parallel computing for our pilot study in Section 4. Below, we describe the setup of the DAP-component as implemented in ActivitySim. Recall that the goal of the DAP-component is to make choices concerning the main activity purposes of persons. For each person, there are three alternatives, namely staying home (H), a mandatory activity such as school or work (M), or a non-mandatory activity such as shopping (N). Since there is dependence between persons within households, the component incorporates the following steps per household.

1. First, the importance rank of each person in the household is determined based on the persons' attributes. This is done to determine the exact dependence between persons of a household, as will become clear in step 3.
2. Then, the utility of all alternatives (H, M, or N) for each person in the household is determined using (1). The attributes of this function include age, income, etc.
3. Subsequently, the utility for all alternatives within a complete household is determined, where importance ranks are taken into account. For a three-person household, one of these alternatives for example is 'HMN', where the most important person stays home, the next important person has a mandatory activity purpose, etc. The utility function on a household level again has the form of (1), where the utilities computed in the previous step act as attributes.
4. Sample daily activity purpose choices for all of the households, using random number generation in combination with (2).

3. GPU-based parallel computation of an ABM's components

Based on the ideas of the previous section, we explain now in more detail how GPU-based parallel computing can be utilized and implemented for the DAP component. It is worth to note, however, that computations of the three other components consist of similar steps to those of the DAP component, and hence all of them are amenable to parallel computing, increasing its potential of reducing computation times.

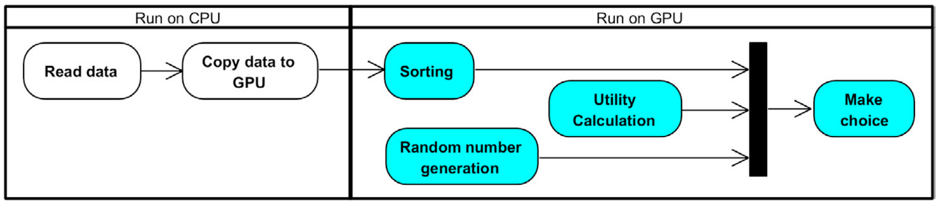

Fig. 1. The non-parallel (left block) and parallel stages (right block) of a component's computations.

It is clear that the above-mentioned steps of the DAP involve computations which are mutually independent between households. Therefore, these steps can be performed simultaneously for different households. Figure 1 shows the computational stages that are involved in the DAP-component. The left block shows the stages which are not parallelized and thus run on the CPU. This includes reading data needed for the computation, such as the attributes of every household and its members corresponding to the C_j in (1), as well as the alternative-specific constants (α_i) and the coefficients ($\beta_{i,j}$). Note that part of the attributes C_j incorporate output of the long-term choice component preceding the DAP-component. This data is then copied from the computer's RAM to the GPU's memory.

The right block, in contrast with the left block, involves stages that are run by the GPU and thus contain the parallel computations. The first of the DAP-component's steps mentioned in Section 2 requires sorting of importance levels. The second and the third steps entail the calculation of each alternative's utilities using (1), and the fourth step involves random number generation in combination with the calculation of (2). We now elaborate on how each of these computational tasks can be handled in parallel fashion using the CUDA framework. The sorting can be done efficiently since CUDA exploits the shared memory in the GPU. Then, the utility function in (1) is calculated by multiplying attribute values (the C_j) with their corresponding coefficients (the $\beta_{i,j}$) and adding a constant (α_i). As (1) needs to be calculated for every alternative and every chooser, the required computations brought by (2) can be interpreted as a matrix-matrix multiplication. Such matrix operations can be done efficiently in parallel using the CUDA library cuBLAS [13]. Likewise, random numbers can be generated efficiently by use of the cuRAND library [14]. Finally, for each chooser, the alternative choice is made by computing (2) for every alternative and subsequent sampling using a generated random number. These computations can be done in parallel quite naturally (and hence need no specific library) in case the alternative choices of the choosers are uncorrelated. When a household is the chooser, this is always the case in the DAP-component.

As mentioned, all parallel computations mentioned above are implemented using the CUDA architecture. This architecture facilitates efficient parallel computation by using the structure of arrays/vectors to store data [15].

As the most time-consuming computations are now done parallelly, we anticipate significant speed-ups in computation time. In the next section, we investigate how large these speed-ups can be.

4. Pilot study and resulting speed-ups

We now perform a pilot study, which compares the computation times of a DAP-component implementation using GPU-based parallel computing by way of the CUDA architecture as explained in Section 3, with those of a similar implementation using the CPU (which is optimized for the CPU) and the implementation of ActivitySim. Both of the latter two components do not incorporate parallel computing. These comparisons are done for different numbers of population sizes, attributes and alternatives in Sections 4.1, 4.2 and 4.3, respectively, to get a feel of their impact on the computation time. The computation time of the GPU implementation includes the time of data transfer between the RAM and the GPU memory, but excludes the time of reading data into the host memory since this time is the same for both the GPU and the CPU implementation.

The computation times of each of the implementations are based on two different datasets, which can be downloaded from [16]. Dataset 1 contains 369845 households with 797674 persons and thus has an average household size of 2.15. We also consider a larger dataset; dataset 2 contains 2732722 households with 7053334 persons and an average household size of 2.58. For each dataset, we use ActivitySim to export partial test sets containing different numbers of households, so that we can report computation times for differently sized scenarios. For each of these scenarios, we present the speed-ups in computation time of the GPU-based implementation as compared to the CPU-

Table 1. Computation time comparisons for dataset 1 when varying the number of households. Asim, CPU and GPU refer to the computation times of ActivitySim's DAP-implementation, our CPU-based DAP-implementation and our GPU-based DAP-implementation, respectively.

Number of households	Population size	Asim [*seconds*]	CPU [*seconds*]	GPU [*seconds*]	Speed-up GPU w.r.t Asim	Speed-up GPU w.r.t. CPU
150000	323911	14.29	2.64	0.48	29.7	5.5
300000	646920	19.33	5.07	0.56	34.7	9.1
369845	797674	21.69	6.12	0.59	36.4	10.3

Table 2. Computation time comparisons for dataset 2 when varying the number of households.

Number of households	Population size	Asim [*seconds*]	CPU [*seconds*]	GPU [*seconds*]	Speed-up GPU w.r.t. Asim	Speed-up GPU w.r.t. CPU
150000	387268	15.81	3.44	0.52	30.5	6.6
300000	774027	22.38	6.79	0.59	37.8	11.5
369845	953841	25.53	8.27	0.63	40.8	13.2
1000000	2578064	55.70	22.52	0.98	56.6	22.9
2000000	5162014	110.57	45.20	1.50	73.7	30.1
2732722	7053334	150.43	61.79	1.77	85.2	35.0

based implementation and the ActivitySim-implementation. All computation times and speed-ups presented in this section are based on computations run on a Windows desktop PC with an Intel Core i9-7900X 3.30GHz CPU, 32 GB RAM, and a NVIDIA GeForce GTX 1080Ti GPU with 3584 CUDA cores and 11 GB memory. In particular, for each computation time or speed-up presented, we ran the underlying actual scenario four times, and took the average of the four computation times.

We have chosen to compare our GPU-based implementation not only with a similarly-coded CPU-based (non-parallel) implementation, but also with the implementation of ActivitySim. We have done this for reference purposes. It is worth emphasizing, however, that the ActivitySim package is written in Python, whereas our GPU and CPU-based implementations are written in C/C++. Python is a programming language that is interpreted, while C/C++ is compiled. This creates additional speed-up effects which cannot be attributed to the introduction of parallel computing. The speed-up of the GPU-based implementation with respect to the CPU-based implementation, however, can be fully attributed to parallel computing.

4.1. *Speed-ups for different population sizes*

We first present the sensitivity of the speed-up to the actual population size of a scenario, which is highly correlated with the number of households. As we can see in Tables 1 and 2 for each dataset respectively, the speed-up of the GPU-based implementation as compared to the CPU based implementation is sensitive to the population size. For small populations, the speed-up is not very high as the overhead of the left block in Figure 1 is then significant. For larger populations, though, the parallelisation capabilities of the GPU can be used to its full potential, as more computations can be done in parallel. We reach speed-up values up to 35 for the DAP-component, from which we conclude that GPU-based parallel computing has a profound potential in ABM. It is interesting to note that the speed-up with respect to the ActivitySim implementation is also increasing. The difference in computation time between the ActivitySim implementation (up to 2.5 minutes) and the study referred to in Section 1 (about 4 hours) is due to the fact that we only consider the DAP-component in this section rather than a complete ABM implementation.

4.2. *Speed-ups for different numbers of attributes*

We now consider the speed-ups when varying the number of attributes (N in Equation (1)) considered in the utility functions underlying the DAP component, based on dataset 2 with 2732722 households. Table 3 shows results for this dataset using utility functions with the original number of attributes (1x), as well as twice and thrice this number of attributes (2x and 3x). When the number of attributes becomes higher, the speed-up of the GPU-implementation over the CPU-implementation hardly increases. This is because parallel computing allows for the concurrent evaluation of different utilities (cf. Equation (1)), but the evaluation of a single utility itself cannot be parallelized efficiently. The increasingness of the speed-up factor with respect to the ActivitySim implementation is an effect that cannot be attributed to the incorporation of parallel computing.

Table 3. Computation time comparisons for different numbers of attributes.

Number of households	Population size	Number of attributes	Asim [*seconds*]	CPU [*seconds*]	GPU [*seconds*]	Speed-up GPU w.r.t. Asim	Speed-up GPU w.r.t. CPU
2732722	7053334	1x	150.43	61.79	1.77	85.2	35.0
2732722	7053334	2x	171.50	63.29	1.83	94.0	34.7
2732722	7053334	3x	189.85	65.45	1.87	101.7	35.1

4.3. Speed-ups for different numbers of alternatives

Table 4 shows the speed-up factors that we obtain when varying the number of alternatives. The computation times reported are based on the same dataset 2 used in Section 4.2. We vary the number of alternatives in the second step of the DAP-component (cf. Section 2) between 3 and 5. This means that for a household with e.g. five members, the number of different alternatives on a household level (i.e. the third step of the DAP-component) varies between $3^5 = 243$ and $5^5 = 3125$. We deduce that also when increasing the number of alternatives, the speed-up of the GPU-based implementation with respect to the CPU-based implementation increases significantly, up to a factor of 50, again marking the potential of GPU-based parallel computing. This is explained by the fact that increasing the number of alternatives also increases the number of utilities that need to be calculated, which can be done in parallel. It is interesting to note that the speed-up with respect to the ActivitySim implementation increases significantly when the number of alternatives become larger.

Table 4. Computation time comparisons for different numbers of alternatives.

Number of households	Population size	Number of alternatives	Asim [*seconds*]	CPU [*seconds*]	GPU [*seconds*]	Speed-up GPU w.r.t. Asim	Speed-up GPU w.r.t. CPU
2732722	7053334	3	150.43	61.79	1.77	85.2	35.0
2732722	7053334	4	185.09	77.60	2.04	90.8	38.1
2732722	7053334	5	319.59	109.82	2.28	140.3	48.2

5. Conclusion and future research opportunities

In this paper, we introduced parallel computing using a computer's GPU to reduce the computation time of ABMs. We implemented GPU-based parallel computing into the DAP-component, and compared its computation time to those of other implementations that do not perform parallel computations. We observed that speed-up factors of up to 50 can be obtained, and that they are highly sensitive to the number of households and the number of alternatives contained in the scenario. We conclude that GPU-based parallel computation addresses the problem of high computations times to a large extent.

The observed speed-up enables the use of ABMs for optimization purposes as well as extensive scenario analysis, which opens up an area of further research. For example, we plan to explore the impact of new smart mobility concepts such as autonomous vehicles and mobility as a service (MaaS) in future work. These concepts introduce more interaction between individuals, which brings the challenge of sharing information among threads in a parallel implementation. Another option is to include additional information in a scenario, such as the weather condition.

Acknowledgements

This research was supported by the NWO Gravitation programme NETWORKS (grant number 024.002.003) and Dutch Applied-Science Organization (TNO).

References

[1] L. Zhang, D. Levinson, Agent-based approach to travel demand modeling: Exploratory analysis, Transportation Research Record: Journal of the Transportation Research Board (1898) (2004) 28–36.

[2] C. R. Bhat, F. S. Koppelman, Activity-Based Modeling of Travel Demand, Springer US, Boston, MA, 35–61, 1999.

[3] J. Castiglione, M. Bradley, J. Gliebe, Activity-Based Travel Demand Models: A Primer, SHRP 2 Report S2-C46-RR-1, 2015.

[4] T. Arentze, H. Timmermans, Multistate supernetwork approach to modelling multi-activity, multimodal trip chains, International Journal of Geographical Information Science 18 (7) (2004) 631–651.

[5] M. Heinrichs, M. Behrisch, J. Erdmann, Just do it! Combining agent-based travel demand models with queue based-traffic flow models, Procedia Computer Science 130 (2018) 858 – 864.

[6] M. A. Bradley, J. L. Bowman, T. K. Lawton, A Comparison of Sample Enumeration and Stochastic Microsimulation for Application of Tour-Based and Activity-Based Travel Demand Models, Paper presented at the European transport conference, Cambridge UK, 1999.

[7] M. Kwak, T. Arentze, E. de Romph, S. Rasouli, Activity-based dynamic traffic modeling: Influence of population sampling fraction size on simulation error, in: International Association of Travel Behavior Research Conference, 1–17, 2012.

[8] M. Saleem, O. B. Västberg, A. Karlström, An Activity Based Demand Model for Large Scale Simulations, Procedia Computer Science 130 (2018) 920 – 925.

[9] K. Wang, Z. Shen, A GPU-Based Parallel Genetic Algorithm for Generating Daily Activity Plans, IEEE Transactions on Intelligent Transportation Systems 13 (3) (2012) 1474–1480.

[10] D. Strippgen, K. Nagel, Using Common Graphics Hardware for Multi-agent Traffic Simulation with CUDA, in: Proceedings of the 2nd International Conference on Simulation Tools and Techniques, Simutools, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 62:1–62:8, 2009.

[11] NVIDIA, CUDA C programming guide, URL https://docs.nvidia.com/cuda/cuda-c-programming-guide/, retrieved on November 22, 2018, 2007.

[12] ActivitySim, An open platform for activity-based travel modeling, URL https://activitysim.github.io/activitysim/index.html, retrieved on November 22, 2018, 2015.

[13] NVIDIA, cuBlas :: CUDA toolkit document, URL https://docs.nvidia.com/cuda/cublas/index.html, retrieved on November 22, 2018, 2017.

[14] NVIDIA, cuRAND :: CUDA toolkit document, URL https://docs.nvidia.com/cuda/curand/index.html/, retrieved on November 22, 2018, 2017.

[15] M. Harris, CUDA workshop pre-ISC2008, Tech. Rep., Dresden, 2008.

[16] Metropolitan Transportation Commission, ActivitySim Powered by Box, URL https://mtcdrive.app.box.com/v/activitysim, retrieved on November 22, 2018, 2016.