A Massive Multilevel-parallel Microscopic Traffic Simulator with Gridlock Detection and Solving

Alex - Alexandru SIROMASCENKO, Ion LUNGU

Economic Informatics and Cybernetics Department, Academy of Economic Studies,
Calea Dorobanți, 15-17, Bucharest, 010552, Romania
alex.siromascenko@gmail.com, ion.lungu@ie.ase.ro

Abstract: Traffic simulators based on microscopic models are a detailed approach to infrastructure and policy evaluation. They allow a closer to reality representation of the factors that influence traffic flow: individual driver and vehicle attributes, dynamic route decisions, lane changing and restrictions, driver cooperation. Such aspects add to the complexity and volume of computations, leading to slower simulation speeds compared to macroscopic models. Also, route restrictions can lead to gridlocks, a common problem in such simulations. In this paper, we propose a multi-level parallel architecture for the TrafficWeb microscopic traffic simulator. The solution combines random load allocation, for multi-threaded processing, and distributed parallelization, through geographical domain decomposition. Adaptive load balancing is used for optimizing the distributed processing speed. Gridlock detection and solving are employed through efficient parallel and distributed algorithms, significantly decreasing their cost. Performance tests show an overall efficiency of 85% for the multilevel-parallel architecture, on a cluster with 5 nodes, each having 4 cores. This allows simulating metropolitan traffic 85 times faster than in real time.

Keywords: Parallel computing, Distributed computing, Multilevel parallelism, traffic simulation, gridlock.

1. Introduction

Traffic modeling and simulation are used in the field of traffic management for evaluating the impact of various road traffic policies and infrastructure changes. Macroscopic models describe the traffic flow attributes of links and intersections, without allowing for the representation of individual vehicles [12]. The key attributes used in macroscopic models are: speed (distance travelled during a time unit), density (number of vehicles in a road segment) and flow (number of vehicles passing through a certain point)[12][13]. Their advantages are computational simplicity and fast simulation speeds. In microscopic models, each vehicle is represented individually and traffic conditions arise as a consequence of vehicle interactions, closer to real life[14].These models are based on the acceleration function, with inputs such as the distance to the vehicle in front, adjacent lane vehicles[12]or even psychological factors[15].Individual driver modeling allows for dynamic route choices, which also impact the overall traffic conditions. Mesoscopic models combine these characteristics, by including individual vehicles and routes. The movement of vehicles on a segment is represented by a queue [7][16], with segment travelling times being approximations derived from macroscopic traffic conditions. Mesoscopic models have recently been used as a solution for simulating nationwide traffic in agent-based systems [12] [17]. Microscopic models are traditionally used for the representation of small, isolated areas, such as a few neighboring intersections, or a city district [12]. With the advent of new technologies for parallel computing, such as multi-core processors and fast network communications, research has been done on large-scale microscopic simulations[6][8].Obtaining fast simulation speeds with microscopic models would allow analyzing metropolitan-scale transportation scenarios, with fewer simulation fidelity compromises.

When designing a traffic simulator, the Real-Time-Ratio (RTR) [16] must be minimized: $RTR = \frac{t_s}{t_p}$, where t_s is the duration of the simulated events and t_p is the time it takes to simulate them. Recently proposed microscopic traffic simulators allow for RTR values of 7.5 [6], 2.5 [1] and 1.5 [5]. These values are estimates for a single-core 2.5 Ghz processor, 100,000 vehicles and $\Delta_t = 1\ sec$, based on the reported performance numbers. The simulator described in [6] can be accelerated through parallel computing, while the other two are not parallel. Among commercial products, VISSIM, AIMSUN, MITSIM, MAS-T2er Lab and ITSUMO are parallel, and only PARAMICS supports distributed processing [10].Its reported RTR is 3.6 using 32 old-generation compute-nodes [24], but constrained by hardware. An older proposed distributed microsimulator [8] has an RTR value of about 48 for 16 CPUs, but without simulating any vehicles.

In microscopic simulators, time is discrete and its granularity is usually given by $\Delta_t = 1\ sec$, the average driver reaction time. Spatially, microscopic simulations are divided into space-discrete and space-continuous. A simulator that conforms to discrete space and time rules falls into the cellular automata category [6], [18], and allows for fast simulation speeds.

2. Proposed Microscopic Simulator

We introduce the microscopic traffic simulator TrafficWeb, designed as a component of an Intelligent Transportation System (ITS). The system's functionalities include predicting and optimizing road traffic across a metropolitan region. Research in this field has seen active support in recent years [21]. The prediction capabilities can be useful by integrating with existing trip planning solutions, like the one described in [22].Agent-based optimization is usually performed over long periods of time, sometimes a few weeks [17], so high RTR values are a requirement.

Being designed for city traffic, the simulator supports lane changing and restrictions, priority rules, traffic lights, and driver cooperation. The last feature is necessary for optimizing correct car placement on mandatory lanes at intersections. TrafficWeb also features the first WebGL-based GUI for a traffic simulator. This allows remote synchronization between a web-based 3D-accelerated graphical interface, and a number of several simulations running in parallel.

TrafficWeb conforms to cellular automata rules. The road network is divided into distinct spaces, called cells, and time is discrete [18]. The maximum cell size that allows realistic traffic representation is equal to the space occupied by a stopped vehicle [6], [18].

According to the principles of cellular automata [6], the simulation is divided into steps and sub-steps, as shown in Figure 1.

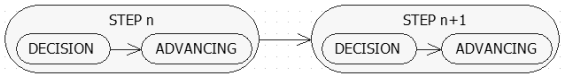

Figure 1. Simulation step/sub-step outline

In the decision sub-step, for each vehicle, the simulator performs all the operations required to determine its new position for the current step, based on information from its vicinity. In the advancing sub-step, the results of the decision sub-step are made visible for all the vehicles. The new position and vehicle states (speed, signaling) will be available as inputs, in the next decision sub-step, for neighboring vehicles. The following simulation constants can be tuned in order to ensure speed-precision scalability:

cl - the size of a cell, the minimum distance a vehicle can advance during a time step

Δ_t- the duration of a time step

V_{MAX} - maximum speed (number of advanced cells between two successive steps) for a vehicle ($\frac{V_{MAX}*cl}{\Delta_t}$).

$AC, DC = \frac{\Delta_V}{\Delta_t}$ maximum speed increase/decrease between two successive steps ($\frac{AC*cl}{{\Delta_t}^2}, \frac{-DC*cl}{{\Delta_t}^2}$)

The main constraint for a realistic traffic simulation is the content uniqueness rule, which requires a cell to be occupied by at most one vehicle during a given time step *t*. The decision rules are defined using the following variables, for each vehicle:

v_{t-1}, v_t - the speed of the current vehicle during the previous and current time steps

d_{t-1} - the distance of the current vehicle to the closest front obstacle, in the previous step

v'_{t-1} - the speed of the closest front obstacle, either another vehicle ($v' \geq 0$) or a stop/slow traffic sign or traffic light ($v' = 0$).

ssp_t - the safe speed for the current time step and vehicle, which allows for safe stopping in any possible situation, current or future

$md(v)$ - minimum stopping distance ($* cl$)

The forward movement rules are computed as follows:

$$md(v) = [v/DC] * (v - DC * \frac{[v/DC]+1}{2}) \quad (1)$$

$$ssp_t = \max(v | (v + md(v)) \leq (d_{t-1} + md(v'_{t-1}))) \quad (2)$$

$$v_t = \begin{cases} \min(V_{MAX}, v_{t-1} + AC), ssp_t \geq v_{t-1} \\ \max(ssp_t, v_{t-1} - DC), ssp_t < v_{t-1} \end{cases} \quad (3)$$

A value of$v_t > ssp_t$ may cause the violation of the content uniqueness rule, as the vehicle cannot stop safely under the max DCconstraint.

The maximum depth (MD) that must be parsed during the decision sub-step, in front of the vehicle, is the one that allows a vehicle to travel at V_{MAX}:

$$MD = \max(d_t) = V_{MAX} + md(V_{MAX}) \quad (4)$$

TrafficWeb supports lane-changing moves, allowing for realistic multi-lane traffic flows. The main rule for lane changing is the clearance rule: the movement of a vehicle$v$ to an adjacent lane at step$t$ must not invalidate the d''_{t-1} value for a vehicle v'' travelling on the target lane, in the vicinity and behind of v. The theoretical maximum clearance for v'' (the maximum depth for the "rear-view-mirror check") is also equal to MD. Finally, MD is also the maximum depth for "priority announcements" at intersections. Figure 2 shows a graphical representation of some of the aforementioned variables:

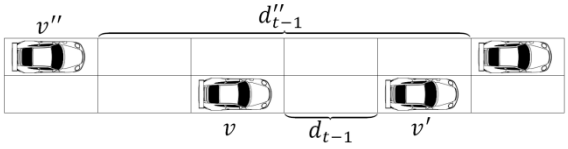

Figure 2. Atypical scenario illustrating safety variables used in the decision sub-step for vehicle v

While microscopic models can include some stochastic parameters [6][18], representing variations in driver behavior, these have been omitted so far, in order to verify the consistency of the results.

Figure 3 shows that the flow of traffic in TrafficWeb is within the limits described in theoretical models. The differences are caused by traffic variations, which are a characteristic of metropolitan environments: downstream jams, traffic lights, lane restrictions.

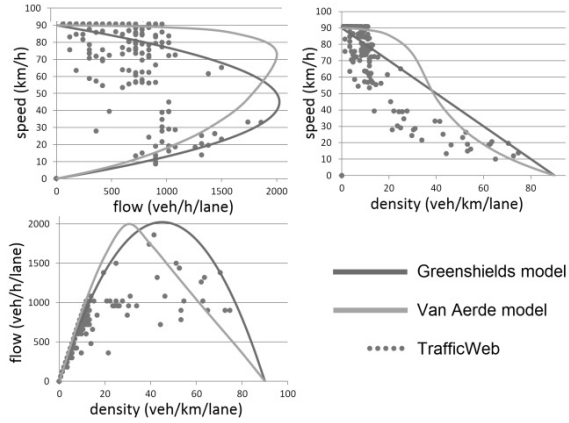

Figure 3. Comparison between TrafficWeb traffic flows and theoretical models described in [13]

3. Gridlock Detection and Solving

In road traffic, gridlock situations occur when certain vehicles cannot advance by applying the safe distance rule, and remain stopped for an infinite amount of time. The main cause of this situation is, as described for a vehicle v:

$n_t(v) = content(advCell(v))$, where $advCell(v)$is the next cell on the route of v

$\underbrace{n_t(n_t(\dots n_t(v)))}_{n\ times} = N_{n_t}(v) = v \Rightarrow v$ is in a gridlock of size n.

Various solutions have been used for managing the gridlock problem. A vehicle is assumed to be part of a gridlock when it has been standing for longer than a threshold period [1], [19], or some undisclosed algorithms are employed for detecting the gridlock [5]. Solving methods include: teleporting vehicles to the next segment [1], removing vehicles from the simulation [19], rerouting vehicles [5], exceeding the estimated segment capacity in mesoscopic models [2] [19], or simulating driver anticipation for avoiding such situations, in more detailed simulators [3].

For solving the gridlock problem, we propose a solution that includes actual detection of the vehicle queue loop and two solving strategies. In order to detect a loop, the vehicle queue is parsed recursively as long as $N_t(v) = v'$ (we haven't reached the end of the queue). A gridlock (G_i) is detected when$v' = v$. Also, a situation of "gridlock dependence" can occur if

$$N_{m_t}(v) = v' \text{ and } N_{n_t}(v') = v' \quad (5)$$

with n as the gridlock length and m as the dependence length. Vehicles in the dependence are not inside the gridlock, but are dependent on its solving to advance. This situation is detected by maintaining a list of all the visited vehicles situated at the start of segments, stopping when such a vehicle is visited twice:

$seg_t(v)$ - current segment of v

$step_t(v)$ - current segment step of v

$SV_{G_i} = \{v \,|N_t(v) = v\,,\ step_t(v) = 1\}$ - start segment vehicles in gridlock G_i.

Gridlocks can be detected during a separate sub-step before the decision sub-step. The front queue is parsed for each vehicle in the simulation. Once a gridlock is detected, the vehicles inside it are marked, and there are two options of solving it.

The first solving method involves granting a "wildcard", for advancing, to each vehicle within the gridlock. This overrides the content uniqueness rule during the decision sub-step,

but the rule is respected in the advancing sub-step, as all the vehicles in the gridlock advance. Each vehicle with a wildcard is allowed to move to the cell in front of it, which is occupied by a blocking vehicle. As the blocking vehicle also receives a wildcard, the aforementioned cell will be freed. In the next step, the gridlock queue will be composed of the same vehicles, each of them shifted forward by one cell. The difference will consist in the dependencies between vehicles: in the new step, vehicles situated at the segments' ends might have the opportunity to leave the gridlock, according to their paths. This cyclic movement is suitable for a simulation involving autonomous vehicles, on which research has increased in recent years [5].

Another approach, suitable for human driver modeling, is the rerouting of end-segment vehicles, shown in red in Figure 4. As in real-life traffic involving human drivers, a driver situated at the end of a segment might get tired of waiting, so he will change his course. The new course will have the same endpoint as the old one and will start at the driver's current position. The restriction for applying this method is that, for a given end-segment vehicle, its current intersection must offer a choice of changing its initial outgoing segment. Thus, the node must have at least two outgoing segments. Also, the newly chosen outgoing segment must allow the driver to reach his intended destination.

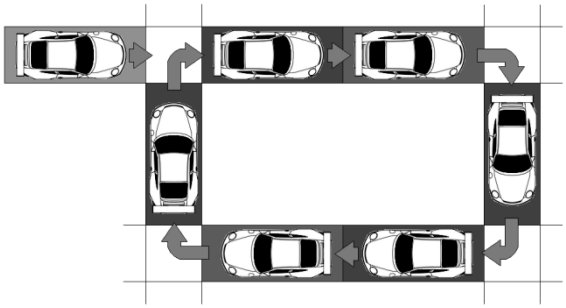

Figure 4. An example situation with gridlock dependence vehicles (orange), start segment vehicles (blue) and other gridlock vehicles (red)

4. Parallel Simulator Architecture

The nature of the traffic simulation problem allows for parallelization. Reduced-complexity queue-based simulators are easier to parallelize, with implementations on SIMD architectures like GPUs [25] resulting in high RTR values [7]. Detailed microsimulations, while difficult to implement on SIMD systems, are highly parallelizable on MIMD architectures [8], [6]. Theoretically, each vehicle may be assigned to a different computation unit, with synchronization required only for time stepping. In TrafficWeb, parallelization is employed at two levels.

4.1 Multi-threaded parallelization

This method is used when the computational load is shared between multiple processing units addressing the same memory space. In recent years, multi-core processors have become the standard in PCs and the trend is expected to grow [9]. Space continuous models require partitioning by lanes among different threads [11]. Other proposed parallel implementations of discrete space models also use geographical domain decomposition [8], [11].

In our proposed architecture, the vehicles that must be processed during the current time step are split among separate threads. We take advantage of the direct memory mapping for individual cells, which allows concurrent markings on the network, without any geographical domain decomposition. This is an advantage when lots of processing units are present, in a shared-memory environment, as there are no overheads associated to partitioning and load balancing. When a vehicle enters the simulation, it is randomly assigned to a thread. For large numbers of vehicles, the thread load is balanced by the random allocation. The lack of explicit load balancing operations compensates for the small random imbalances that may occur. A few principles ensure a good scalability of the multi-threaded simulation. First, the absence of per-vehicle synchronizations reduces computationally expensive OS calls. As shown in Figure 5, thread waiting and notification operations are used only at the step / sub-step levels. Second, using visible uncertainty reduction, the results of certain queries, like front vehicle determination and gridlock inclusion, are shared among threads during the decision sub-step.

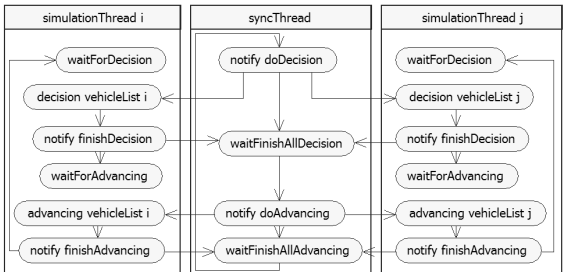

Figure 5. Thread synchronization in the multi-threaded parallel architecture

To accelerate the gridlock detection, it is performed during the decision sub-step, as opposed to a separate sub-step. The gridlock state $gs(v)$of each vehicle v, once determined, is visible from the other vehicles' perspectives (including those handled by other threads), in the decision sub-step:

$gs(v) \in \{0,1,2\}$ – state codes for *unknown*, *in gridlock, not in gridlock*

$gs(v) = 2$, if $\exists\, v' = N_t(v),\ s(v') = 2$

$gs(v) = 1$, if $\exists\, v' \in SV_{G_i}$, with $seg_t(v') = seg_t(v)$and $N_t(v') = v$

This allows for a parallelization of the gridlock detection algorithm, using multi-threading.

4.2 Distributed parallelization

This method is employed by processing the simulation on independent systems, which communicate through a network interface. A number of several workstations is common in most computing environments, so further speedups can be obtained by processing on a Beowulf cluster. Gigabit Ethernet is standard nowadays, but its speed is significantly lower (1 Gbit/s) compared to RAM (>85 Gbit/s in DDR3) [4]. Thus, the random assignment of vehicles among separate nodes is not efficient. Due to the geographical nature of the problem, spatial decomposition [2], [8], [11], [20] can be used for limiting the amount of data transferred between nodes. This is achieved by assigning different contiguous areas of the map to different nodes [8], [11], [20]. Finding optimal partitions requires solving the graph partitioning problem, with two objectives: minimizing communication volumes between neighboring partitions, by limiting the inter-partition visibility [8], [20];ensuring a balanced load distribution for all nodes, in this case a balanced number of vehicles, during each iteration [20], [24]. The graph partitioning problem is known to be NP-hard [8], without known algorithms for finding the best solution. A fast approximate solution can be obtained using orthogonal bisection [8], [20]. For low inter-node message volumes, only information about vehicles that may influence the neighboring partition's vehicles is sent. The presence of vehicle v on the network may influence other drivers' behaviors at a maximum distance of MD, both upstream and downstream. There are three types of visibilities, as shown in Figure 6:

$VU_{pp'} = \{c \in p \mid \exists\, c' \in p', d(c,c') < MD\}$ - upstream visibility

$VD_{pp'} = \{c \in p \mid \exists\, c' \in p', d(c',c\,) < MD\}$ - downstream visibility

$VP_{pp'} = \{c \in p \mid \exists\, c' \in p',\ d(c, dn(s')\,) < MD\}$ – priority visibility, where $dn(s')$ is the destination node of segment s'

$V_{pp'} = VS_{pp'} \cup VD_{pp'} \cup VP_{pp'}$ contains all cells from partition p visible to partition p'

Figure 6. Areas of length MD (darker),from the red partition, which are visible to the green partition

$M_{pp'_t} = \{A_{v_t} | cell_t(v) \in V_{PP'}\}$is the message from p to p' during time step t ,with A_{v_t} the collection of attributes (24 bytes) for vehicle v at time t.

For each partitioning, the visibility ratio $vr = \frac{vc}{C}$ (visible cells relative to the total number of cells) is an approximation of the expected message exchange ratio. Figure 7 shows a graphical representation of this ratio.

Figure 7. Segments with visible cells (black) in an orthogonal bisection partitioning of Bucharest

In the distributed simulation algorithm, synchronization between nodes is required before the beginning of each step. Each vehicle is handled by only one node during a given decision sub-step. For each visible vehicle, 9 values, totaling 24 bytes, are passed over the network, to the neighboring partitions. This includes information about vehicle id, the newly computed speed and position, as well as signaling and inter-

vehicle cooperation values. As shown in Figure 8, the visible-vehicles information is transmitted between nodes after the decision sub-step. When this information arrives, the receiving node performs the advancing sub-step for these vehicles, adopting them if their new position is in the node's partition.

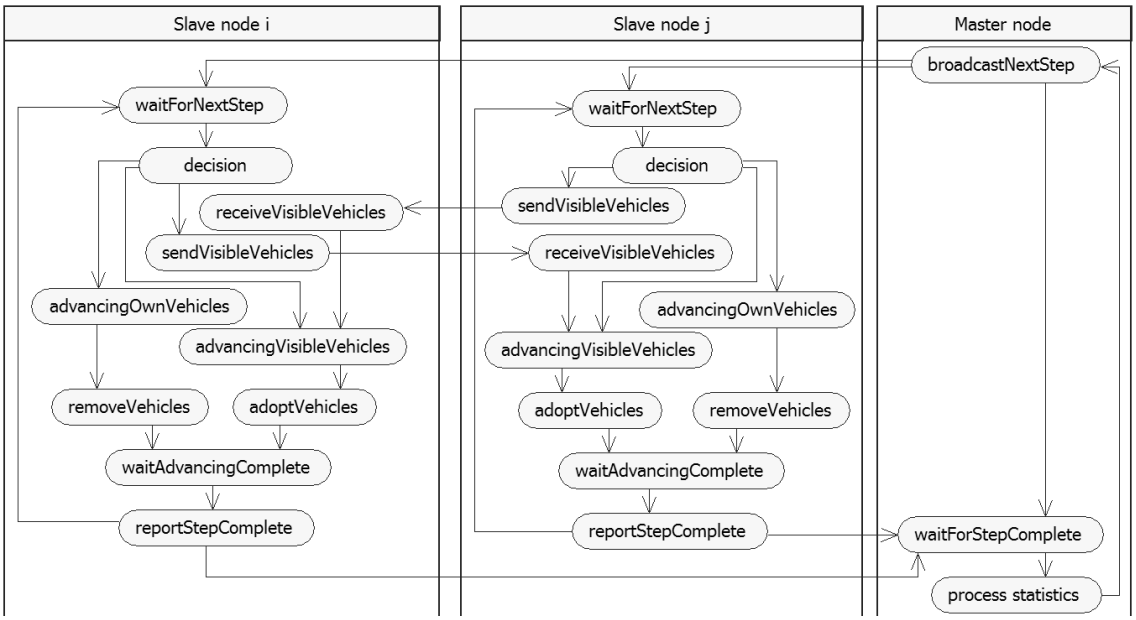

Figure 8. Outline of message passing in the distributed parallel architecture

Communication between adjacent partitions is asynchronous with the other computations within a given simulation step. The only constraint is that any advancing of vehicles (own or visible) in a node must not begin before the decision sub-step for the given node has ended. The advancing of own / visible vehicles runs multi-threaded and in parallel.

5. Distributed Gridlock Detection

In the distributed simulation, gridlock detection is achieved by allowing the loop to be reconstructed, based on information from each partition's edges. For any given partition p, the following potential-gridlock visibilities are determined:

$$VGD_{p_t} = \{v' \in p' | \exists\, v \in p, n_t(v) = v'\}$$

$$VGU_{p_t} = \{v \in p | \exists v' \in p', n_t(v') = v\}$$

$$VG_{p_t} = \{r = (v, v') | v \in VGU_{p_t}, \; v' \in VGD_{p_t}, \; N_t(v) = v'\}$$

$VG_r = \{v'' \in p |\, N_t(v) = v''; \; N_t(v'') = v'\}$ for a given $r \in VG_{p_t}$

Globally, cross-partition queue edges are in

$$VG_t = \bigcup_{p \in P} VG_{p_t}$$

We define the function $nr: VG_t \to VG_t$

$$nr(r) = nr\big((v, v')\big) = (v', v'') \in VG_t$$

At a given time step t, $v \in p$ is in reconstructed gridlock if

$$\exists r \in VG_{p_t}; v \in VG_r; \; nr(nr(\dots(nr(r)\big)) = r$$

6. Adaptive Load Balancing

During a simulation, the load distribution on the network may vary greatly. This leads to load imbalances and wasted computing resources. Also, as the nodes may have heterogeneous processing capabilities, assigning an equal number of vehicles to each node might not be optimal. In order to efficiently use all the available resources, we use a load balancing mechanism. The mechanism takes into account both variations in spatial load distribution and the heterogeneity/variation of the available computing resources. As shown in Figure 9, the master node checks for discrepancies in computing times for each slave node. The time cost of the decision sub-step is analyzed as a benchmark for each node, during each simulation step.

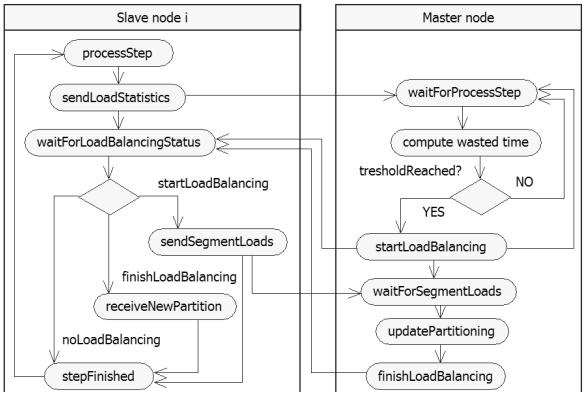

Figure 9. Outline of the load balancing algorithm

$C_{n_t} = \frac{card(V_{n_t})}{DTn_t}$ - the processing capacity of node$n$at step t, as $vehicles/ms$

$C_t = \sum_{n \in N} C_{n_t}$ - cluster processing capacity

$V_t = \bigcup_{n \in N} V_{n_t}$ - all the vehicles that must be processed on the cluster

$IL_{n_t} = card(V_t) * C_{n_t}/C_t$ - the ideal vehicle load for node$n$ at step t

$IT_t = IL_{n_t}/C_{n_t}$ - the ideal processing time, constant at step t, $\forall\, n \in N$

$OTn_t = DTn_t - IT_t$ - overload time for n

$WT_t = \max_{n \in N}(OTn_t)$ - global wasted time

$IR_t = WT_t/IT_t$ - the imbalance ratio at step t

Load balancing is performed depending on the value of θ (treshold), when $IR_t > \theta$.

The load balancing is considered finished when all the slave nodes have received their updated partition segment lists. After repartitioning run r, each partition p receives information MP_{p_r}:

S_{p_r}- segments in p at run r

$RS_{p_r} = \left\{ s' \in S - S_{p_r} \middle| \exists s \in S_{p_r}, d(s,s') \leq V_{MAX} \right\}$-segments of other partitions, reachable in one time step from p

$d(s,s')$- distance from s to s'

$dn(s)$- destination node of segment s

$p_r(s)$- partition of$s$in run r

$$VU_{p_r} = \{ s' \in S - (S_{p_r} \cup RS_{p_r}) \mid \exists\, s \in (S_{p_r} \cup RS_{p_r}), d(s,s') \leq MD \vee d(s',s) \leq MD \vee d(s, dn(s')) \leq MD \}$$

$$MP_{p_r} = \left\{ s \in \left(S_{p_r} \cup RS_{p_r} \cup VU_{p_r} \cup MP_{p_{r-1}} \right) \middle| (s \notin MP_{p_{r-1}}) \vee (p_r(s) \neq p_{r-1}(s)) \right\}$$

For the partitioning run r, partition p receives information only about the segments concerning it, for which it doesn't have information from the previous partitioning run $r-1$, or which changed their partition from $r-1$ to r.

7. Experimental Results

For simulation tests, the road network of the city of Bucharest was used, consisting of 52,782 segments and 24,953 nodes. The data was imported from OpenStreetMap using a purpose-built XML parser and was persisted in an Oracle Spatial database.

The simulation was calibrated using the following parameter values:

$cl = 450\ cm, V_{MAX} = 4, AC = 1, DC = 2$

A variable number of 25,000 to 100,000 simultaneously starting trips was employed, as described in Table 1. The value of 100,000 is a typical load value for the road network of Bucharest [23].

Table 1. The scenarios used for testing

No.	Start veh.	End veh	Average veh.
1	25000	20000	23255
2	50000	30000	34728
3	100000	80000	90603

The trips were previously obtained by randomly generating pairs of origin-destination nodes, with each path computed using Dijkstra's algorithm, having segment length as link cost. This way, congestion formed along the main boulevards.

Table 2 shows the number of gridlocks for the tested scenarios. This is greater than expected to occur in real life, but microsimulations are known to over represent such situations. The number of gridlocks increases with the number of vehicles, due to greater congestion. All the gridlocks were successfully solved using cyclic movements.

Table 2. Description of gridlock situations

Vehicles	Gridlocks	Max gridlock size
25,000	31	61
50,000	96	442
100,000	628	680

Figure 10 shows that gridlock searching has a significant impact on the simulation time, increasing it by 38% for 100,000 vehicles. The parallel search algorithm greatly decreases this performance penalty, down to 11% when using 8 threads, a decrease of 72%. The speedup for the gridlock searching algorithm is thus 3.5.

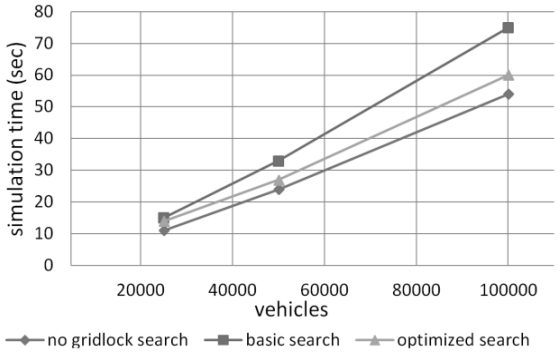

Figure 10. Performance impact of gridlock searching

For the tested scenario, the vr is 0.34% for 2 partitions, 0.8% for 5, 2.05%for 16 and 19.5% for 1024 partitions. Figure 11 shows an increase of the visibility ratio, at a decreasing rate, when the number of partitions grows. The actual number of visible vehicles is greater than the estimate because both vehicles and partition visibility areas tend to concentrate in the central area of the city.

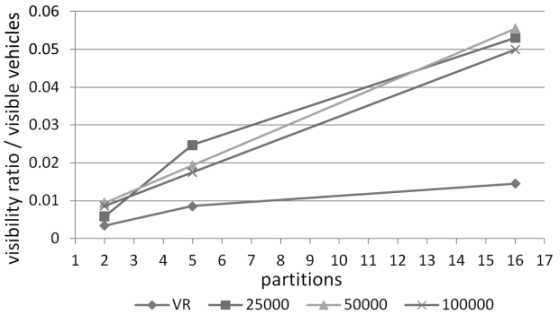

Figure 11. Evolution of the vr estimate and actual visible vehicles ratio

The processing cluster was composed of the nodes described in Table 3. All the nodes featured 4 physical cores and 8 virtual cores, through Hyper-Threading, and 8 GB of RAM.

Table 3. Specifications of the processing cluster

Node	Processor	C_n(vehicles/ms)
$n1$	Core i7 2670QM 2.2GHz	3292
$n2$	Core i7 2670QM 2.2GHz	3285
$n3$	Core i7 2630QM 2GHz	3087
$n4$	Core i7 930 2.8GHz	3039
$n5$	Core i7 Q7401.73 GHz	2782

The Java socket bandwidth was benchmarked and was found to be at 88% of the theoretical maximum (1Gbit/sec). This value allows sending information for 4583 vehicles/ms across the cluster.

Figure 12 shows that the speedups increase when the problem size (number of vehicles) grows, due to the diminishing impact ratio of the synchronization operations. The multi-threaded (MT) performance scalability was tested on$n1$, and its maximum speedup is 3.5 using 4 physical cores and 4.15 using 8 virtual cores. Hyper-Threading thus brings a speedup of 18%.For the distributed (DT) scalability, the base value (1 node) is the one obtained using only $n5$, and all the nodes were assigned equally sized partitions. In the 100,000 scenario, distributed processing on 5 nodes speeds up the simulation by 4.45, equivalent to an efficiency of 89%.

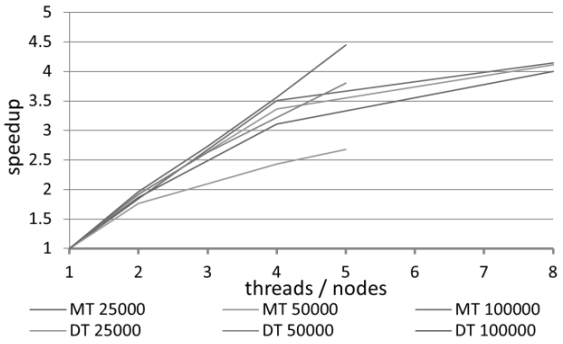

Figure 12. Speedups obtained using multi-threading (MT) and distributed (DT) processing

Figure 13 shows that gains in RTR value are higher when simulating more vehicles. With adaptive load balancing (LB values), the RTR is increased by 11%.

Figure 14 shows the overall increase in the value of RTR, using the multilevel parallel architecture, with and without adaptive load balancing. Processing on multiple cores increases the RTR value from 5 to 21 when

using node n_1, and, by adding 4 additional nodes to the cluster, the value increases to 87.

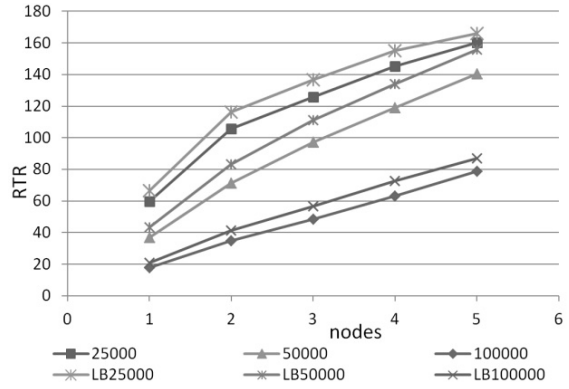

Figure 13. The increase of the RTR value using distributed processing and load balancing

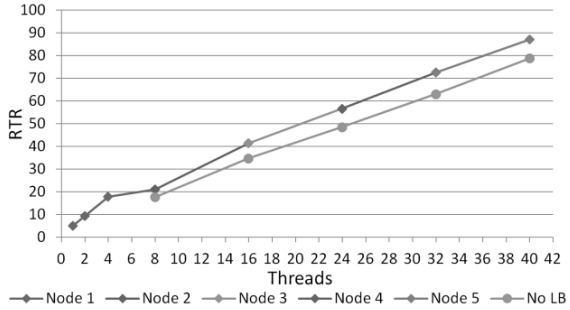

Figure 14. RTR values for the overall number of used threads in the cluster, 100,000 vehicles

The overall performance speedup obtained using multilevel parallel processing is 17.17. The overall efficiency of the distributed system is thus 85%, if taking into account the theoretical maximum imposed by the number of physical cores:

$$4\ cores * 5\ nodes = 20x\ speedup$$

8. Conclusion and Future Work

The proposed multi-level parallel architecture efficiently accelerates computations in a metropolitan-scale traffic microsimulation, resulting in high RTR values. Multi-level parallelism is an advantage, with random load allocation at node level being more efficient than further splitting each node into sub-partitions for each thread. Such smaller partitions would be more prone to short-term imbalance variations. With the highest obtained RTR value of 87, a week of traffic in the city of Bucharest, averaging 90,000 vehicles at any time, can be simulated in only 1h and 55min. The overall multi-level parallel efficiency is 85% using 5 nodes. Performance is improved by using a processing-capacity aware load balancing algorithm for the cluster environment, as all the available resources are optimally used. The load balancing mechanism adapts to both differences in computing capacities and variable vehicle load distributions across the network. This can be further evaluated using a more pronounced variation of the congestion areas.

The gridlock detection algorithm is optimized for parallel processing, reducing its impact on the overall processing time. The distributed detection method allows the simulation to run continuously in a distributed environment.

The TrafficWEB platform will be extended by including continuous trip scheduling and vehicle insertion, and also dynamic traffic assignment. Thanks to the high RTR value, the evaluation of various algorithms for dynamic traffic allocation, traffic signaling and lane restrictions can be performed in a reasonable amount of time.

Acknowledgements

This work was co-financed from the European Social Fund through Sectorial Operational Programme Human Resources Development 2007-2013; project number POSDRU/ 107/1.5/S/77213 „Ph.D. for a career in interdisciplinary economic research at the European standards”

REFERENCES

1. ***Sumo wiki - http://sumo.sourceforge.net/doc/current/docs/userdoc, Nov. 2011
2. WARAICH, R., D. CHARYPAR, M. BALMER, K. AXHAUSEN, **Performance Improvements for Large Scale Transportation Simulation in MATSim**, 9th Swiss Transport Research Conference, Sep. 2009.
3. DONIEC, A., R. MANDIAU, S. PIECHOWAIK, S. ESPIÉ, **A Behavioral Multi-agent Model for Road Traffic Simulation**, Engineering Applications of Artificial Intelligence, vol. 21 Issue 8, December, 2008, pp. 1443-1454.
4. ***List of device bit rates - http://en.wikipedia.org/wiki/List_of_device_bit_rates
5. CARLINO, D., M. DEPINET, P. KHANDELWAL, P. STONE, **Approximately Orchestrated Routing and Transportation Analyzer: Large-**

scale Traffic Simulation for Autonomous Vehicles, 15th International IEEE Conference on Intelligent Transportation Systems, 2012, pp. 334-339.

6. KORCEK, P., L. SEKANINA, O. FUCIK, **A Scalable Cellular Automata based Microscopic Traffic Simulation**, Intelligent Vehicles Symposium (IV), 2011. IEEE Transcriptions.
7. STRIPPGEN, D., K. NAGEL, **Multi-agent Traffic Simulation with CUDA**, Intl. Conference on High Performance Computing & Simulation, 2009. HPCS '09.
8. NAGEL, K., M. RICKERT, **Parallel Implementation of the TRANSIMS Micro-simulation**, Parallel Computing, vol. 27(12), November 2001, pp. 1161-1639.
9. LIN, P. **Desktop and Notebook PC Technology Penetration Forecast**, iSuppli Market Intelligence, Nov. 2012.
10. PASSOS, L. S., R. J. F. ROSSETTI, **Towards the Next-generation Traffic Simulation Tools: a First Appraisal**, Information Systems and Technologies (CISTI), 2011 6th Iberian Conference.
11. ANNING, N. I., J. ZHICAI, Y. JIAONI, **Method and Strategy for Parallelizing microscopic Traffic Simulation**, Proc. of the 10th Intl. Conf. of Chinese Transportation Professionals, 2010.
12. BARCELÓ, J., (Ed.) **Fundamentals of Traffic Simulation**, vol. 145, Springer, 2010
13. RAKHA, H., B. CROWTHER, **A Comparison of the Green shields, Pipes, and Van Aerde Car-Following and Traffic Stream Models**, Transportation Research Record: Journal of the Transportation Research Board, vol. 1802, 2002, pp. 248-262.
14. EHLERT, P., L. ROTHKRANTZ., **Microscopic Traffic Simulation with Reactive Driving Agents**, IEEE Proc., Intelligent Transportation Systems, 2001.
15. BRACKSTONE, M., M. MCDONALD, **Car-following: a Historical Review**. Transportation Research Part F: Traffic Psychology and Behaviour vol. 2(4), 1999, pp. 181-196.
16. NAGEL, K., K. MARCHAL, **Computational Methods for Multi-agent Simulations of Travel Behavior**. Proc. of the Meeting of the Intl. Assoc. for Travel Behavior Research (IATBR), 2003.
17. MEISTER, K., *et al*., **Large-scale Agent-based Travel Demand Optimization Applied to Switzerland, Including Mode Choice**. ETH, Eidgenössische Technische Hochschule Zürich, IVT, Institut für Verkehrsplanung und Transportsysteme, 2010.
18. NAGEL, K., M. SCHRECKENBERG, **A Cellular Automaton Model for Freeway Traffic**, Journal de Physique vol. I 2(12), 1992, pp. 2221-2229.
19. RIESER, M., **Adding Transit to an Agent-Based Transportation Simulation: Concepts and Implementation**. PhD thesis, VSP, TU Berlin, Germany, 2010.
20. ÇETIN, N., **Large-scale Parallel Graph-based Simulations**. Dissertation, ETH Zurich, Switzerland, 2005
21. BANCIU, D., I. PETRE, D. M. SMADA, M. ANGHEL, **Developing an Interactive System to Provide Management Support for Transportation Research Organizations**, Studies in Informatics and Control, vol. 20(4), 2011, pp. 420-428. ISSN 1220-1766.
22. HRIN, G. R., L. E. ANGHEL, M. TOMESCU, I. ILIESCU, D. SAVU, **Solutions for Finding the Optimum Route between Two Urban Locations Using Public or Private Transport or Pedestrian Movement**, Studies in Informatics and Control, vol. 17(4), 2008, pp. 353-360, ISSN 1220-1766.
23. WSP Group Romania, **Master Plan General pentru Transport Urban – Bucuresti, Sibiu si Ploiesti** - Raport Final Bucuresti, EuropeAid/123579/D/SER/RO, 2007.
24. LIU, H. X., *et al*., **Distributed Modeling Framework for Large-scale Microscopic Traffic Simulation**, Proc. of 84th Annual Meeting of the Transportation Research Board (CDROM), 2005.
25. LUNGU, I., A. PÎRJAN, D. PETROŞANU, **Optimizing the Computation of Eigenvalues using Graphics Processing Units**, Scientific Bulletin Series A, Applied Mathematics and Physics, 2011, ISSN 1223-7027.