Acceleration of Traffic Simulation on Reconfigurable Hardware

Justin L. Tripp, Henning S. Mortveit, Matthew S. Nassr, Anders A. Hansson, Maya Gokhale
Los Alamos National Laboratory
Los Alamos, NM 87545
Email: {jtripp, henning, mnassr, hansson, maya}@lanl.gov

***Abstract*— Previously, traffic simulation on FPGAs was limited to very short road segments or required a large number of FPGAs. The appearance of large (3 – 10M gate) FPGAs with a large number of input/output pins (up to 1200) allows for large amounts of data to be processed concurrently. This work shows that traffic simulation of entire metropolitan areas is possible with FPGAs using a data streaming approach. This overcomes scaling issues associated with constructive approaches and still allows for high-level parallelism by dividing the data sets across multiple FPGAs. This result paves the way for accelerating other large infrastructure simulations, by modeling the problems with FPGA hardware in mind.**

I. Introduction

Modern society relies on a set of complex, inter-related and inter-dependent infrastructures. Los Alamos National Laboratory has over the past ten years developed a sophisticated simulation suite for simulating various infrastructure components, such as road networks (TRANSIMS [1]), communication networks (AdHopNet [2]), and the spread of disease in human populations (EpiSims [3]). These powerful simulation tools can help for example policy-makers understand and analyze these inter-related systems and support decision-making for better planning, monitoring, and proper response to disruptions. TRANSIMS, for example, can simulate the traffic of entire cities, with people traveling in cars on road networks. It is based on interacting cellular automata (CA), and requires the use of large computer clusters for efficient computation.

In this work we study the acceleration of the road network simulation through an FPGA implementation. Since the simulation is parallel, with independent agents that make decisions based on local knowledge, it seems natural to map to the large-scale spatial parallelism offered by FPGAs. The high degree of regularity found in the road network is another fact making this a well suited application. In contrast, the networks found in the AdHopNet project (telecommunication) are much more irregular.

In the next section we provide an overview of the TRANSIMS software-based road network simulator as well as other efforts of hardware-based simulation of traffic. We then describe two different approaches based on the TRANSIMS microsimulator and their mapping to hardware. Performance results for both implementations on two different Xilinx FPGAs are presented. Finally, we end with conclusions and opportunities for future work.

II. Related Work

TRANSIMS is a tool for large-scale traffic simulation and analysis of entire cities. A short description of how it operates is as follows: First, a synthetic *population* is created based on survey data for the given city. It is created in a such a way that all statistical quantities and averages considered are consistent with the survey data. Examples of such quantities are age distributions, household sizes, income distributions, and car ownership distributions. In the next stage, realistic *travel plans* are made for all the individuals for a twenty-four hour day. An example plan could be 'bring kids to school – go to work – pick up kids – stop at the grocery store – drive home.' The *router* coordinates the plans of all individuals to produce realistic *travel routes* with realistic travel times. The router operates together with the *micro-simulator* which is the module responsible for moving entities around. TRANSIMS uses the actual transportation infrastructure of the city, so a route could look like 'start at A – drive to B – walk to C – take shuttle to D.' Further information can be found at [1] along with descriptions of a recent study of the Portland metro area. Our FPGA implementation will only be concerned with the micro-simulator and will be limited to cars and will not include buses, trains, and so on. The details of the micro-simulator are given in the next section.

The Portland TRANSIMS study will be used as an example of the requirements of a large traffic micro-simulation [4]. The Portland road network has $124,904$ links, with the average link length close to 250 meters. Assuming 1.5 lanes/link on the average and using the TRANSIMS standard 7.5 meter cell length, there are roughly 6.25 million road cells. For cities like Chicago, Houston, and Los Angeles this number is larger by a factor of $3\times$ to $10\times$.

FPGAs have been applied to the traffic simulation problem, but not on a metropolitan scale. The earliest work using Xilinx FPGAs, implements a limited length of single-lane road with intersections [5]. This is used to simulate the load characteristics of a road, based on the available throughput. Their approach uses a constructive approach to link together different types of road blocks. A more recent FPGA based traffic simulator by Bumble, uses a streaming approach based on a discrete event simulation framework [6]. Bumble implements a discrete event simulator in FPGAs and uses this to simulate the traffic interactions. His road models are limited to single-lanes with simple intersections. To implement a single four-way intersection using this approach requires 30–32 Altera Apex FPGAs.

Our work differs from previous approaches in two important ways. First, FPGAs are now much larger and more capable than before. Second, our approach attempts to extend simulation beyond a few roads and intersections to whole metropolitan areas. Previously, the cost of FPGA traffic simulations at the metropolitan scale was too expensive. Here, we determine whether FPGAs can be used to efficiently simulate whole cities.

III. CA Traffic Modeling

The TRANSIMS micro-simulator can best be described as a cellular automaton computation on a semi-regular grid or cell network: The city road network is split into nodes and links. Nodes correspond to locations where there is a change in the road network such as an intersection or a lane merging point. Nodes are connected by links that consist of one or more unidirectional lanes. Each lane is divided into road cells. The TRANSIMS road cell is 7.5 meters long. One cell can hold at most one car. A car travels with velocity $v \in \{0, 1, 2, 3, 4, 5\}$ cells per iteration step. The positions of the cars are updated once every iteration step using a synchronous update. The maximal speed of a car is cell dependent, but it is at most 5. An iteration step advances global time by one second. The basic driving rules for multi-lane traffic in TRANSIMS consists of four steps. In each step we consider a single cell i in a given lane and link. Note that our model allows passing on the left and the right. To avoid cars merging into the same lane, cars may only change lane to left on odd time steps and only change lane to the right on even time steps. This convention, along with the four driving rules desribed below, produces realistic traffic flows as demonstrated by TRANSIMS.

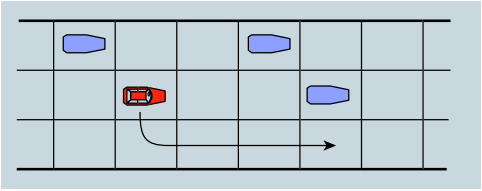


Fig. 1. CA traffic in TRANSIMS

A. Local driving rules

The four basic driving rules of the micro-simulator are given in the following. We let $\Delta(i)$ and $\delta(i)$ denote the cell gap in front of cell i and behind cell i, respectively.

1) *Lane Change Decision:* Odd time t: If cell i has a car and a left lane change is *desirable* (car can go faster in target lane) and *permissible* (there is space for a safe lane change) flag the car/cell for a left lane change. The case of even numbered time steps is analogous. If the cell is empty nothing is done.
2) *Lane Change:* Odd time t: If there is a car in cell i, and this car is flagged for a left lane change then clear cell i. Otherwise, if there is no car in cell i and if the right neighbor of cell i is flagged for a left lane change then move the car from the neighbor cell to cell i with probability p_α. The case of even times t is analogous.
3) *Velocity Update:* Each cell i that has a car updates the velocity using the two-step sequence:
   - $v := \min(v + 1, v_{\max}(i), \Delta(i))$ (acceleration)
   - If $[\mathrm{UniformRandom}() < p_{\mathrm{break}}]$ and $[v > 0]$ then $v := v - 1$ (stochastic deceleration).
4) *Position Update:* If there is a car in cell i with velocity $v = 0$, do nothing. If cell i has a car with $v > 0$ then clear cell i. Else, if there is a car $\delta(i)+1$ cells behind cell i and the velocity of this car is $\delta(i) + 1$ then move this car to cell i. The velocity update pass (3) guarantees that there will be no collisions.

All cells in a road network are updated simultaneously. The steps 1–4 are performed for each road cell in the sequence they appear. Each step above is thus a classical

cellular automaton Φ_i. The whole combined update pass is a product CA

$$\Phi = \Phi_4 \circ \Phi_3 \circ \Phi_2 \circ \Phi_1,$$

where product is function composition. Note that the CAs used for the lane change and the velocity update are stochastic CAs. The rationale for having stochastic breaking is that it produces more realistic traffic. The fact that lane changes are done with a certain probability avoids slamming behavior where whole rows of cars change lanes in complete synchrony.

B. Intersections and Global Behavior

The four basic rules handle the case of straight roadways. TRANSIMS uses travel routes to generate realistic traffic from a global point of view. Each traveler or car is assigned a route that it has to follow. Routes mainly affect the dynamics near turn-lanes and before intersections as cars need to get into a lane that will allow them to perform the desired turns.

To incorporate routes the road links need to have IDs assigned to them. Moreover, to keep computations as local as possible, cells need to hold information about the IDs of upcoming left and right turns.

The following describes the extension of the four basic driver rules to handle turn-lanes and intersections.

Modification of the lane change rule:

We consider a car in cell i. As before, lane changes to the left/right are only permissible on odd/even numbered time steps. We refer to the adjacent candidate cell as the target cell.

1) If the link ID of the target cell matches the next leg of the travel route a lane change is desirable (desirable turn-lane).
2) Else, if the target cell has a link ID that does not match the next leg of the route and it differs from the current link ID of the route, a lane change is not desirable (wrong turn).
3) Else, if the current cell's nextLeftLink (nextRightLink) ID matches the next leg of the route and the simulation time is an odd (even) integer, a lane change is desirable (prepare for turn-lane or intersection).
4) Else, apply the basic lane changing rule described above.

Note that this handles lane changing prior to turn-lanes as well as intersections.

Intersection Logic

An intersection has a number of incoming and outgoing links associated to it. A simplified set of turning rules (assuming a four-way intersection) are as follows:

1) Only cars in an incoming left(right)-most lane of link can turn left(right). A car that turns left(right) must initially use the left(right)-most lane of the target link.
2) A car in any incoming lane can go straight. A car that goes straight must use the same lane number in the target link as it used in the incoming link. It is assumed that the lane counts for the relevant links agree.

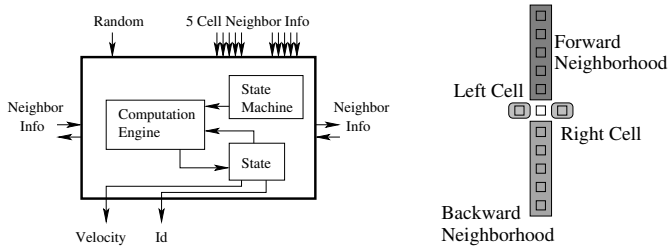
More intricate intersection geometries can of course occur but the basic idea remains the same. When intersecions are close it is natural to modify the first rule: when a left turn is followed by an immediate right turn the rightmost lane is chosen as target lane for the left turn.

An intersection has a set of immediate adjacent road cells. We refer to these as the *intersection road cells*. The intersections operate by dynamically assigning the front and back neighbor cell IDs of the intersection road cells. This allows us to naturally extend the driving rules for multi-lane traffic to intersections without any modifications. The subset of the intersection road cells that come from incoming links have their front neighbor cell set to zero by default. The same holds for the back neighbor of the intersection cells belonging to outgoing links. The intersections operate by establishing front/back pairs between cells to accommodate the routes. Stop intersections and traffic signal intersections impose additional constraints on which cars are allowed to drive at what times by controlling the corresponding connections.

IV. Constructive Approach

The constructive approach to traffic simulation creates a separate simulatable road cell for every road cell in the traffic network. The road cell provides its current state to its neighbors so that all the cells in that local neighborhood can calculate their next state. Figure 2 outlines the structure of a basic road cell.

The road cell consists of three main parts: the computation engine, the state and a state machine. The state machine drives the computation engine using the current state and inputs from external road cells to compute the road cell's next state. The rules described in Section III dictate what operations are included in the computation engine.



(a) Cell Design (b) Neighborhood

Fig. 2. Road Cell Design and Cell Neighborhoods

The four rules for traffic simulation are executed using six different states in the state machine. Figure 3 shows how the different steps in the rules are executed in the state machine. Each rule in the computation engine requires a single cycle to calculate except for *Velocity Update*. The velocity update rule has three separate operations, each taking a cycle, to calculate its two steps.

In the LaneChange state, the computation engine calculates the lane change decision. To do this the $\Delta(i)$ and $\delta(i)$ are calculated from the forward and backward neighborhoods. Likewise the neighbors in the left and right neighborhoods execute the same calculation.[1] The computation engine then determines whether it is permissible for cars to come to this lane and whether the current car desires to change lanes. These results are used in the LaneMove state to actually perform the lane change. Both lanes have to agree that is is both permissible (where we are going) and desirable (if the gap ahead of us is worse than our neighbors) for a lane change to happen.

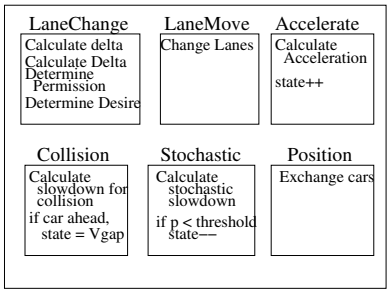


Fig. 3. Computation required for the six states

In the Accelerate state, a car's velocity is calculated using the following formula: $v_{next} = \min(v + 1, v_{max}(i))$. $v_{max}(i)$ is the maximum velocity for this particular road cell, which may be lower than the global v_{max} (e.g., a local speed limit).

The Accelerate state is followed by the Collision state which ensures that the next state does not exceed the gap ahead of the car. It determines $v_{next} = \min(v_{next}, \Delta(i))$. This prevents cars from accelerating into a car in front of them—avoiding a collision.

The final step of the velocity update determines if the car should randomly slow down. This stochastic step provides some realism in the behavior of drivers and makes their speeds less predictable. If a random value is less than a threshold, p_{break}, then its speed will be lowered as described in Section III.

After the velocity update rule is finished, the state machine executes an update of the car positions. To do this, a cell determines if a car exists in its backward neighborhood that has a velocity that will bring it to this cell's location. If it does, then the cell sets its velocity and car id to the arriving car. Otherwise, if no car is arriving at this cell, the cell sets its velocity and car id to zero.

The FPGA implementation is written in VHDL and synthesis was performed by Synplify v7.6. The resultant EDIF description was passed into Xilinx ISE v6.2 to produce the results reported.

The results for the constructive approach for multi- and single-lane circular traffic are described in Table I. The hardware implementation of single-lane traffic has only four states, since single-lanes do not require the extra hardware for lane changes. The two-lane implementation that includes the hardware to perform lane changes is 63% larger in area.

TABLE I
FPGA DESIGN RESULTS

	One-lane		Two-lane	
	XC2V6000	XC2VP100	XC2V6000	XC2VP100
Cells	650	650	400	400
LUTs/Cell	104	97	169	175
Clock(MHz)	48.68	64.17	35.53	54.43
Slices	33790	31576	33790	34999
(% of Slices)	(99%)	(71%)	(99%)	(79%)

Table II compares the results for the two-lane traffic implementation achieved by the Xilinx XC2V6000 and the XC2VP100 to a software implementation running on two different Xeon processors. The speedup reported is relative to the 1.7GHz Xeon. The XC2V6000 simulates the road cells at a rate $303.1\times$ the Xeon. This speedup comes primarily from the fact that the FPGA implementation is executing all cells concurrently, and

[1]These neighborhoods can be reduced by road geometry. For example: a reduction of lanes, dead-ends, intersections, etc.

the software implementation, which may have instruction level parallelism, calculates each cell individually.

TABLE II
RESULTS COMPARISON FOR TWO LANE IMPLEMENTATIONS

	XC2V6000	XC2VP100	1.7GHz Xeon	3.0GHz Xeon
Cells	400	400	600	600
Cells/sec	2.37×10^9	3.64×10^9	7.82×10^6	1.35×10^7
Speedup	303.1	651.1	1.0	1.73

Despite the large speedup that is possible using the constructive approach, the FPGA can only handle a small number of road cells. Using the data from the Portland TRANSIMS study, we know that there are roughly 6.25 million road cells. Simulating Portland would require at least 12,400 FPGAs to simulate the entire city. This limitation in scalability using current technologies led to the development of a streaming approach to calculate the traffic simulation.

V. STREAMING APPROACH

We have seen how the constructive hardware approach does not scale well, because it relies on the creation of a predetermined number of road cells. An alternative approach is to let a computational unit, an *update engine*, process a *stream* of road data and subsequently output a *stream* of updated data. In this way, the update engine sweeps across the road data, and the number of road cells is no longer a memory constraint to the FPGA implementation. Instead, this streaming approach is only limited by the host memory size (which holds information about the road data) and the associated access time, and the hardware design becomes inherently scalable and can thus handle large-scale road networks.

In the streaming approach, we decided to focus the hardware on the straight lane computation, leaving intersections and merging links to be calculated in software. Most importantly, this strategy means that all road plan decisions are handled by the software, and the hardware processing is governed by a simple, homogeneous set of traffic rules.

In a basic implementation of streaming, data representing the road network in the previous state is loaded into the input SRAM from the host. The data is subsequently fed into the update engine against the flow of traffic, starting from the end of the link. In the case shown in Figure 5, there are four lanes of data per link. Each lane has its own input and output SRAM, and each car both changes lanes and moves forward inside the internal logic engine before it is written with its new road cell and new velocity information to its new lane's output SRAM. Since the road cells are processed in order, and one at a time, the address generators are counters (only enabled when valid data has come out of the SRAM on the previous read).

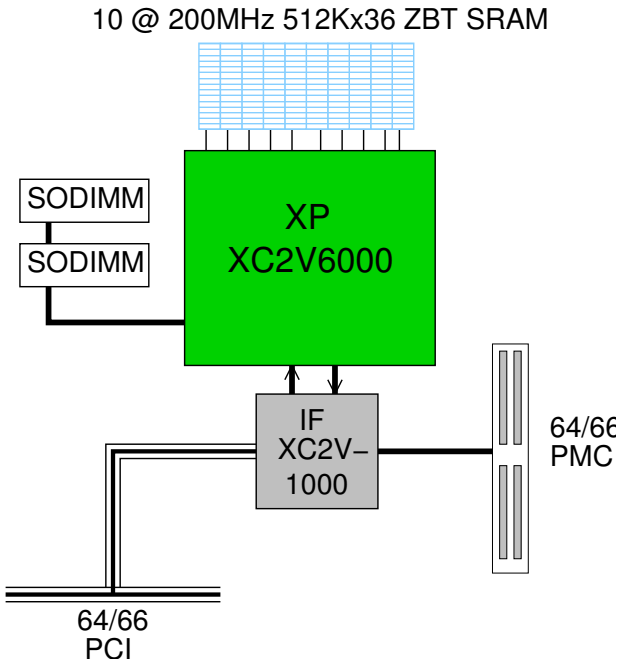


Fig. 4. Structure of the Osiris FPGA Board.

Based on the Portland network system with its 6.25 million road cells, and the fact that the Osiris [7] board (shown in Figure 4) with its ten 512k×36 ZBT SRAMs provides enough room for about 5 million road cells, we have assumed that there will be enough SRAMs available to hold all road data for the straight lane segments (about 75% of the total number of cells) being updated on the FPGA. On the Osiris board we used four of the ten 200MHz 512k×36 ZBT SRAMs which provide enough room for about 2 million road cells, a figure close to half of the Portland road network system. We have not taken into account the data transfer between the FPGA and host necessary for information exchange to and from the software updating the merge nodes and intersections.

Inside the logic engine, the road cells are first scanned for cars. If a car is found, the car must have its position updated based on its attributes from the previous state and the location of the surrounding cars.

The cars in the first v_{max} cell layers of a lane can not be updated (either by changing lanes or by changing velocity) since there is not enough information available. For this reason we define the overlap region as the end

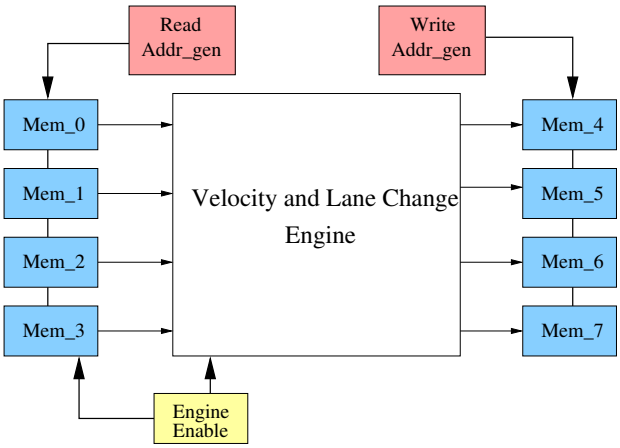


Fig. 5. Structure of a straightforward streaming implementation.

v_{max} cell layers of a link. These cell layers are the first segments processed, and while cars can be moved into these road segments, they cannot be moved from them. The software driven intersection/link-merge updater will update the position and velocity of these cars.

Changing lanes and changing velocity are executed as two separate calculations. Position update must be calculated after changing lanes, since it directly depends on the location of the cars. A pipeline calculating the lane changes and velocity updates allows results to be read and written every clock cycle.

1) **Changing Lanes:** The functionality works exactly as described by the micro-simulator. However, since a car must have complete information about the road segments in front of and behind it to a distance of v_{max}, it is only immediately outside of the overlap region where the cars will have enough information to make a lane change decision. Therefore, for a car to have enough neighbor information to make a lane change decision, the decision must be made v_{max} clock cycles after the car's information has come out of the SRAM. Until then, the car's information moves though a shift register, and its existence is used as neighbor information for the cars ahead.
2) **Changing Position:** To change position, the first step is calculating the new velocity based solely on the car's old velocity, and the existence of any car v_{max} or less in front of the car in its own lane. After the new velocity is calculated, the car does not move to the appropriate road segment immediately. Instead, it moves into a shift register (see Figure 7).

It continues to move one cell per clock cycle through the shift register's lookup blocks (see Figure 6) until the point where its newly calculated velocity matches its distance from the change state block. At this point, the change state block imports all car information from the car designated to be moved out of the lookup buffer to the new destination road segment. This allows for the new destination road segment containing the moved car data to be written into the output SRAM on the next clock edge.

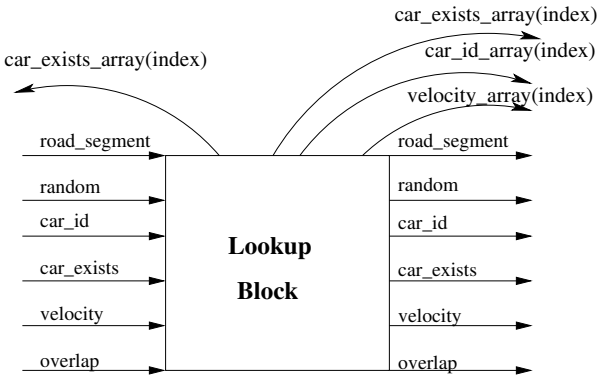


Fig. 6. The Lookup Block

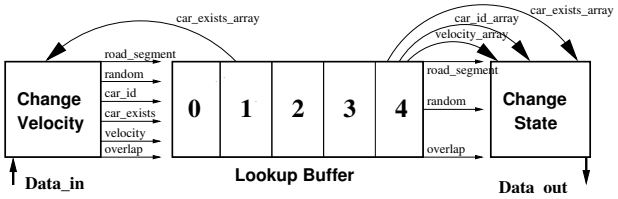


Fig. 7. The Position Update

The above implementation has been found to have the results on XC2V6000 and XC2VP100 boards shown in Table III. These results show that there is a drastic cut in occupied board space with a subsequently higher clock speed. This is to be expected with only about 50 road segments being processed at a time compared to the 400 segments in the constructive approach.

The results for the production of updated road cells is understandably slower with the streaming approach than with the constructive approach as shown in Table II. However, the speedup factor of 30-50 is still a vast improvement over a software implementation.

It can be seen that although the acceleration with the streaming approach decreases by a factor of 10 from the constructive approach, it is still enough of an improvement to provide significant traffic simulation

TABLE III
STREAMING PROCESSING RATES

	XC2V6k	XC2VP100
Clock(MHz)	61.1	96.6
Slices	2167	2095
Slices/Cell	541	523
Occupied Slices (%)	6%	4%

TABLE IV
STREAMING COMPARISON WITH MICROPROCESSORS

	XC2V6k	XC2VP100	1.6GHz Xeon	3.0GHz Xeon
Cells/sec	2.44×10^8	3.86×10^8	7.82×10^6	1.35×10^7
Speedup	31.2×	49.4×	1.0×	1.73×

speedup. Since the streaming approach is inherently scalable, there is no need for more FPGA space for larger-scale simulations. The number of FPGAs will affect the number of update engines which will determine the ultimate number of road segments updated, but we have shown a significant speedup with only one update engine. It is true that with our implementation we are limited by the size of the SRAM on board, but since it is only desirable and not necessary to have all road segment information in the SRAM at all times, this is not a size limiting factor.

The hardware has shown to perform very well with the straight lane segments of the links it was given. It works best with the repetitive, route plan oblivious parts of the traffic simulation that make up 70–90% of the road segments in a given simulation. FPGA aided simulation done in the scalable, streaming approach may be the fastest way to do extremely large metro-area traffic simulations, especially in light of the advances being made in combined microprocessor/FPGA computing systems. The cellular nature of the road segments meshes well with hardware, and a combined hardware/software approach for the full-fledged simulation fits each of their computational strengths.

VI. CONCLUSION

The two approaches, constructive and streaming, have demonstrated two different extremes in scalability. The constructive approach is able take advantage of a high degree of parallelism to obtain a speedup of 300 to 600× over software simulation. However, due to the high cost of FPGA area, this approach is not very cost effective. The streaming approach trades area for time and is able to obtain a 30 to 50× speedup over software. This speedup is still significant and could be increased by adding multiple FPGAs.

Scaling issues have been overcome by using a streaming architecture. However, scaling is not the only problem when simulating an entire city. The traffic simulation accelerator must be integrated into the larger TRANSIMS framework. In its current form, the FPGA implementation handles multi-lane traffic and leaves intersections and route plans to be executed by the CPU.

This partitioned approach could provide a large benefit on freeways, but would not work as well with shorter links. There is also a large amount of per-update communication with the TRANSIMS module that handles nodes and link end-points. Overcoming these communication requirements is a challenge.

Acceleration of TRANSIMS opens the door to a whole range of simulations where FPGAs or other dedicated hardware can provide computational speedup. Many simulations systems today, such as EpiSims, have similar structure to the one found in TRANSIMS: There are highly complex computations best suited for software and a large collection of structured simple calculations as in the micro-simulator. A successful implementation accelerating TRANSIMS will provide a prime example as to how FPGAs can aid a large class of large-scale simulations.

More importantly, this will also help pave the way for a new simulation modeling paradigm. Rather than taking an existing simulation with structure identified as being suited for hardware acceleration, one would now do the problem modeling with the possibilities of hardware in mind. The structural insight and knowledge obtained in this work should provide a good start for this.

REFERENCES

[1] L. L. Smith, "Transims home page," 2002. [Online]. Available: http://transims.tsasa.lanl.gov/

[2] K. A. Atkins, C. L. Barret, R. J. Beckman, S. G. Eubank, N. W. Hengarter, G. Istrate, A. V. S. Kumar, M. V. Marathe, H. S. Mortviet, C. M. Reidys, P. R. Romero, R. A. Pistone, J. P. Smith, P. E. Stretz, C. D. Engelhart, M. Droza, M. M. Morin, S. S. Pathak, S. Zust, and S. S. Ravi, "ADHOPNET: Integrated tools for end-to-end analysis of extremely large next generation telecommunication networks," Los Alamos National Laboratory, Los Alamos, NM, Tech. Rep., 2003.

[3] S. Eubank, H. Guclu, V. S. A. Kumar, M. V.Madhav, A. Srinivasan, Z. Toroczkai, and N. Wang, "Modelling disease outbreaks in realistic urban social networks," *Nature*, vol. 429, no. 6988, pp. 180–184, May 13, 2004.

[4] C. L. Barrett, R. J. Beckman, K. P. Berkbigler, K. R. Bisset, B. W. Bush, K. Campbell, S. Eubank, K. M. Henson, J. M. Hurford, D. A. Kubicek, M. V. Marathe, P. R. Romero, J. P. Smith, L. L. Smith, P. E. Stretz, G. L. Thayer, E. Van Eeckhout, and M. D. Williams,

"TRansportation ANalysis SIMulation system (TRANSIMS) portland study reports," December 2002. [Online]. Available: http://maynard.tsasa.lanl.gov/PortlandStudyReports.html

[5] G. Russell, P. Shaw, and J. McInnes, "Rapid simulation of urban traffic using fpgas," 1994. [Online]. Available: http://citeseer.ist.psu.edu/russell94rapid.html

[6] M. D. Bumble, "A parallel architecture for non-deterministic discrete event simulation," Ph.D. dissertation, Pennsylvania State University, 2001.

[7] B. Schott, P. Bellows, and L. Wang, *Osiris Board Architecture and VHDL Guide*, ISI-East, University of Southern California, Arlington, VA, January 16, 2004, Release 2.3.0.