Large-Scale Parallel Computing Implementation of TRANSIMS Microsimulation Model of the Chicago Metropolitan Area

Michael Hope
Computational Transportation Engineer
Energy Systems Division
Argonne National Laboratory
transims@anl.gov
http://www.tracc.anl.gov/

Introduction to TRANSIMS Microsimulation

- The TRANSIMS microsimulator is an agent-based simulation code which operates well on the meso or micro scale to assess congestion patterns.
- Routing of all the agents takes place in a separate utility run just prior to the microsimulator. This routing details which set of road links will make up the agent's path.
- The agents are then simulated along the fixed paths generated by the router all together in the microsimulator. Vehicles have full discretion on the best way to move through a given link.
- Agents follow a set of rules based on a cellular automata framework with additional behaviors to model:
  - Random slow down
  - Voluntary lane changes
  - Lane swapping
  - Driver reaction mechanics
  - Proper plan following distances

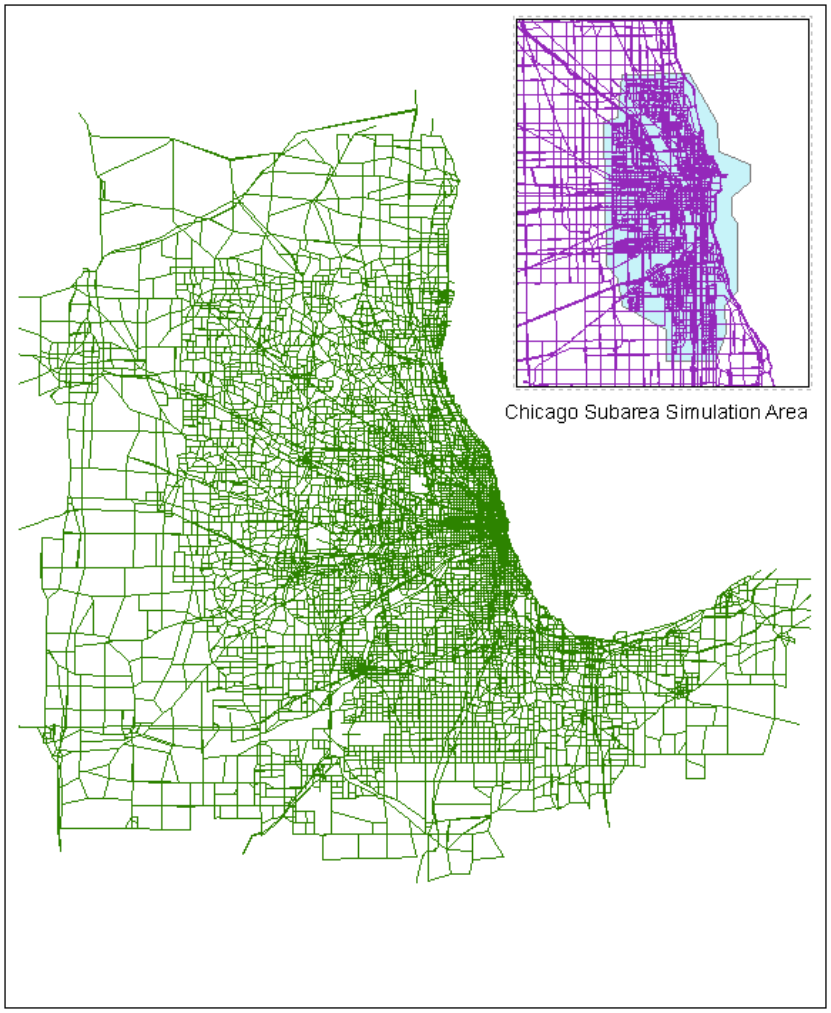
The Evolution of TRANSIMS Microsimulation

- TRANSIMS was originally developed at a time when large scale microsimulation required cluster computing capabilities.
- Version 3 of TRANSIMS was designed to run exclusively across multiple processors on a cluster computer.
- Standard desktop processors are now fast enough to perform microsimulation over reasonably large regions.
- Version 4 was designed to optimize performance for a single core, discarding all elements of the previously parallel framework.
- Three factors dictated the need for TRANSIMS to become a parallel application once again:
  - Desktop processor architecture moved from having a fast single core to having multiple slower cores.
  - Small (12- to 24-core) clusters became widely available.
  - TRANSIMS microsimulation is now being used on areas large enough that current desktop performance is not sufficient in terms of both memory and processing time

The Chicago TRANSIMS model

- The Chicago Metropolitan Area model (developed by TRACC) is an ideal test bed for the parallelization effort of TRANSIMS for several reasons:
  - The Chicago regional model memory requirements are too large to run on a desktop computer: ~27 million trips over a 24 hour period
  - Subarea microsimulation methods with ~3.5 million trips still take over 2 hours and do not reveal high fidelity congestion patterns in the regional network
  - TRACC has full access to a high performance cluster

Chicago Metropolitan Area Regional Network



Parallelization Architecture Overview

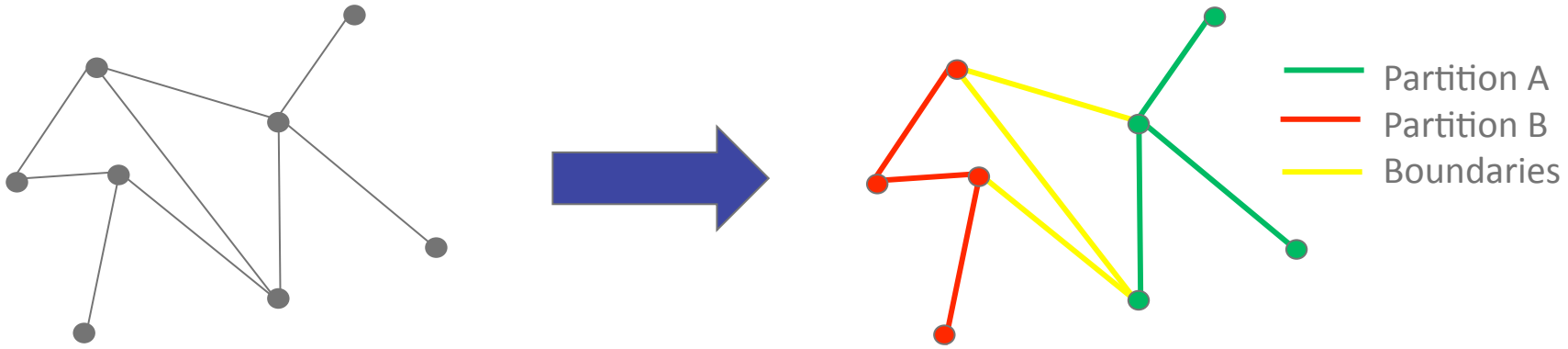
- The parallelization of the TRANSIMS microsimulator was done with several motivations in mind:
  - Produce realistic mechanics across boundaries
  - Minimize pre-processing and post-processing
  - Minimize time needed to process and transfer boundary vehicles
  - Provide a framework which facilitates dynamic re-routing across partitions and allows dynamic load re-balancing during the simulation

Responsibilities of Each Node

- Master Node (1)
  - Reading in trip paths for all partitions
  - Passing trip path out to appropriate slaves
  - Gathering congestion data from slave nodes
  - Writing congestion statistics
  - Global monitoring of activity-based travelers
- Slave Nodes (n-1)
  - Processing second-by-second movements of all vehicles within the geographical area assigned to it
  - Exchanging boundary vehicles with neighboring partitions
  - Writing snapshot output
  - Performing the necessary boundary calculations

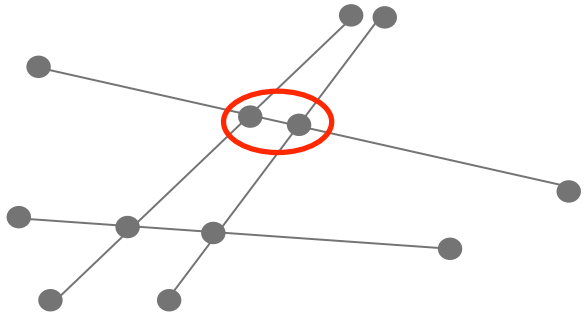
PartitionNet: Overview

- Before simulation the network is first broken into geo-partitioned regions which are then assigned to each processor.
- METIS is a fast open source graph partitioning program written in C which takes an adjacency list of nodes, weights, and desired number of partitions as input. The output is a partitioned graph
- The program PartitionNet was developed as a wrapper around the METIS software to re-format, weight, and export the TRANSIMS road network to METIS. After partitioning, the METIS output is read back in and re-formatted as a TRANSIMS network. Partitions are denoted as an extra entry in the TRANSIMS node table.

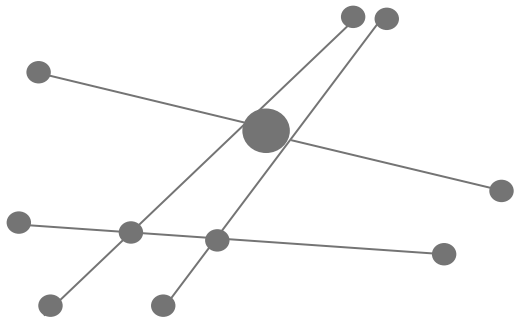


PartitionNet: METIS Preparation Techniques

Determine Short Links



Compress them into one node, aggregating the connectivity of the original nodes



Re-Write Link table as an adjacency list

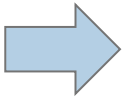
Link	Anode	Bnode
1	1	2
2	2	3
3	3	1

Node	Adjacent Nodes
1	2, 3
2	1, 3
3	1, 2

PartitionNet: METIS Preparation Techniques cont'd

Calculate weights one of three ways:

1) Sum half the capacities of all links associated with the node
2) Sum half the lengths times lanes of all links associated with the node
3) Sum the occupancy values of all links associated with the node

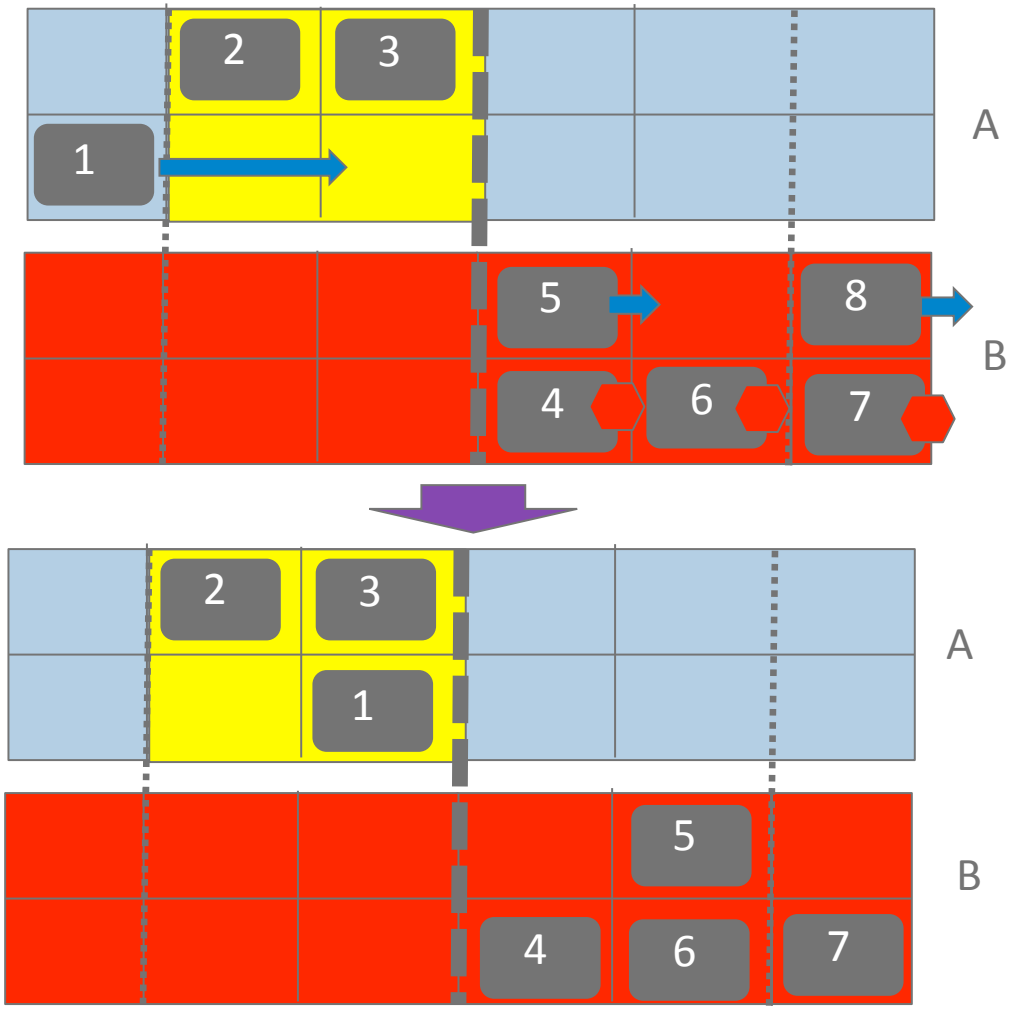


Weight	Node	Adjacent Nodes
8	1	2, 3
10	2	1, 3
2	3	1, 2

Architecture

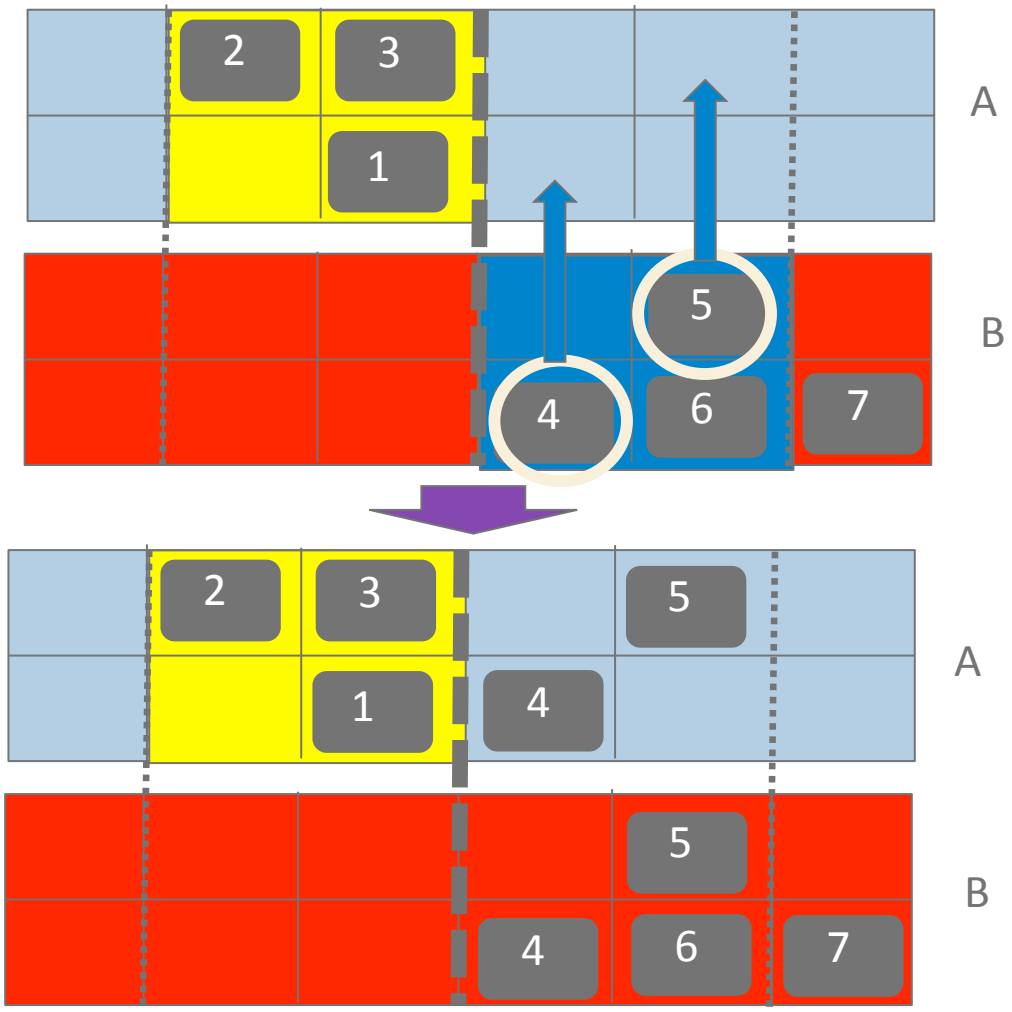
Process first vehicles not in gold boundary region:
1, 4, 5, 6, 7, and 8

Anyone who moves into the gold boundary region is placed in the boundary queue: 1



Architecture cont'd

Determine the trailing vehicles within the blue region and pass them back to partition A

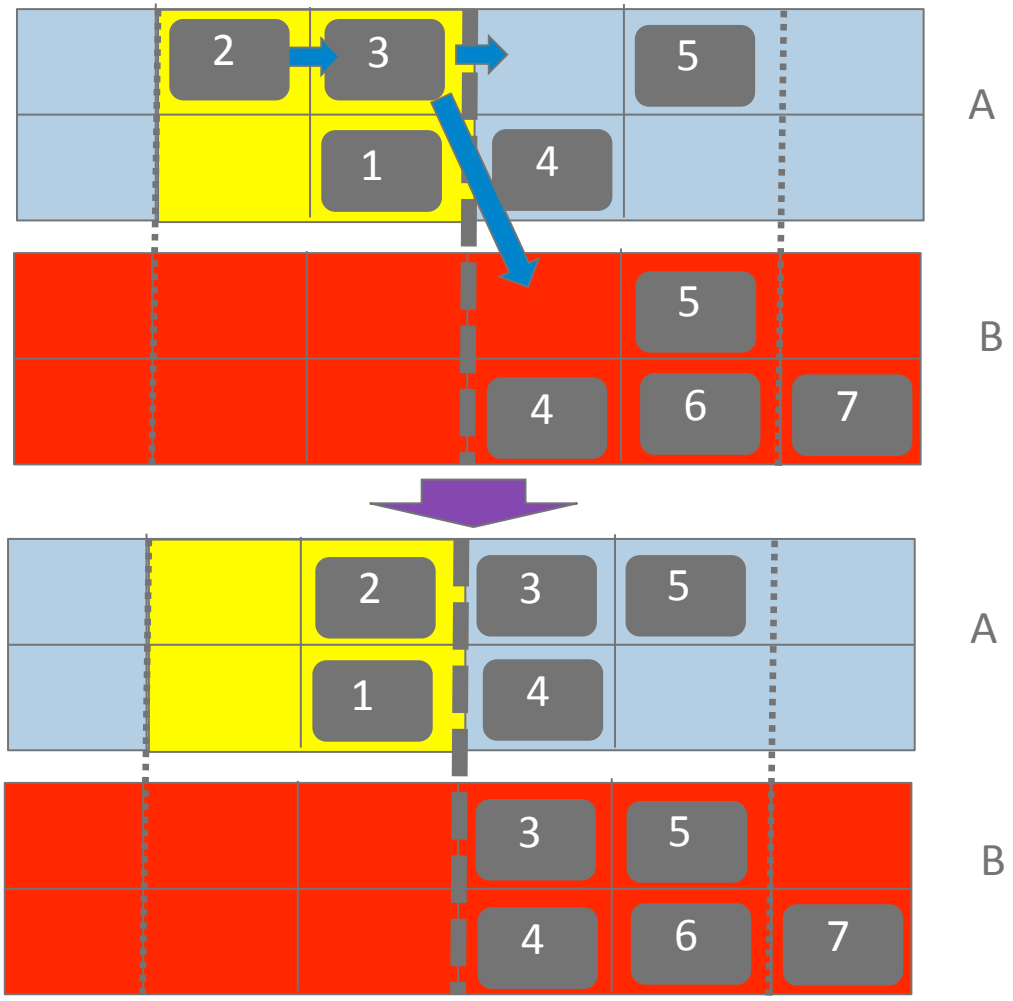


The only information actually passed is the vehicle's position

Architecture cont'd

Process the vehicles in the boundary queue (gold region) which have not yet moved this step: 2, 3

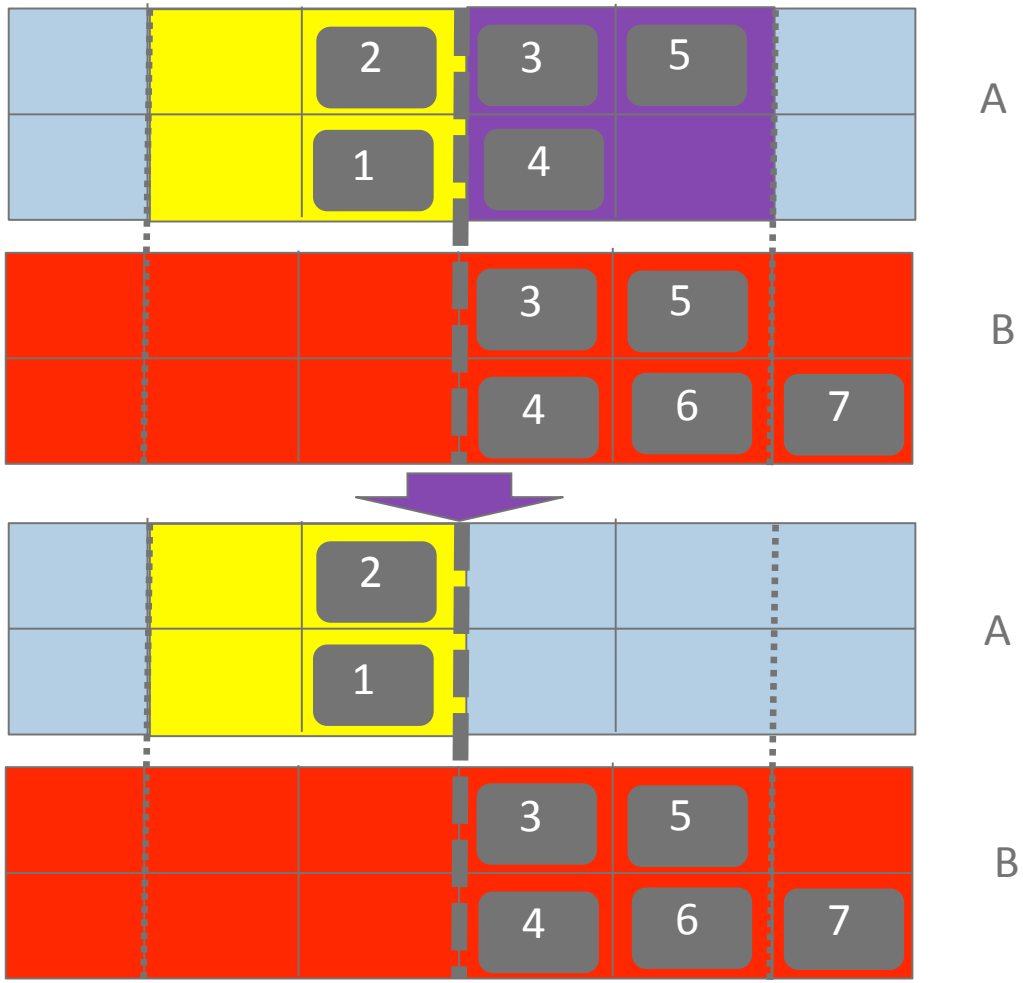
All vehicles which cross the boundary copy and pass all relevant vehicle information (path, ID, position, etc…) to partition B



Architecture cont'd

Remove all vehicles in the purple region of partition A and place any new vehicles in partition B in the priority queue for the next iteration.

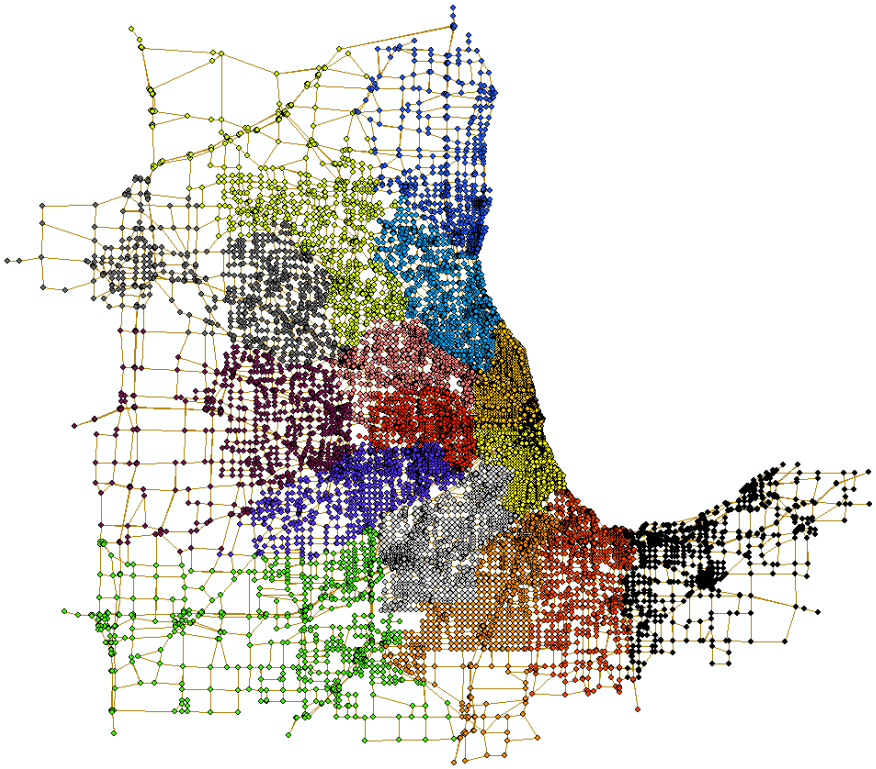
This boundary traversal method minimizes transferred data and has the distinct advantage that every vehicle is still only processed once per iteration.



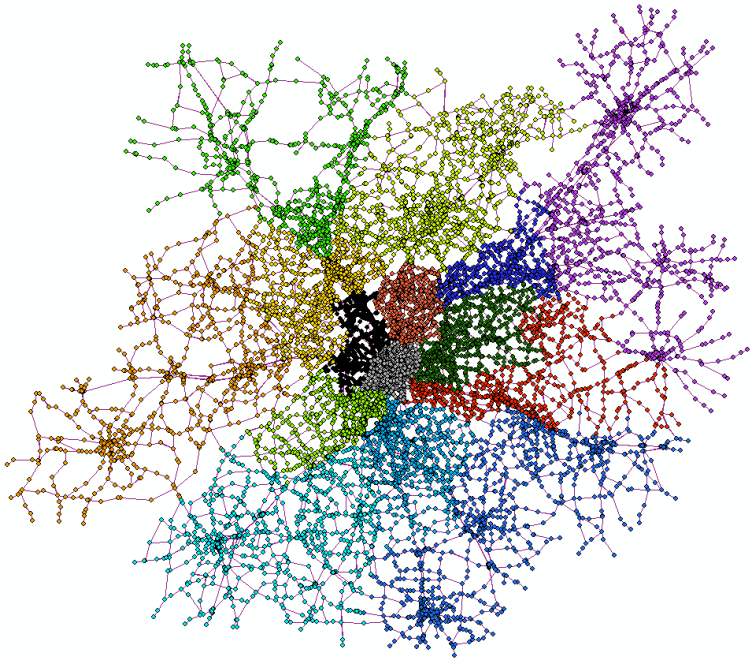
Additional Features

- Full functionality for Transit
  - Massively multi-cell vehicles (trains) move uniquely through boundaries by "guessing" where their cells were in the previous partitions – saving valuable transfer time
  - Small multi-cell (buses) can exist in both partitions at one time
  - Passengers transfer successfully between partitions
- Full functionality for Activity-Based Trips – Activity Monitoring System
  - Assures that travelers won't begin legs of trips in other partitions if they encountered a problem in another partition
  - Travelers will not start a leg of their trip in one partition if they are still busy in another partition
- Various memory-based optimizations
  - Redesign of the vehicle (primary storage) array - saving every byte possible
  - Clearing unneeded parts of the vehicle (primary storage) array dynamically as necessary
  - Ensuring things done on a region-wide basis are only done in the portion of the network important to that partition
  - Detecting and fixing memory leaks using Valgrind

Regional Networks Subdivided into Partitions



Chicago Metropolitan Area

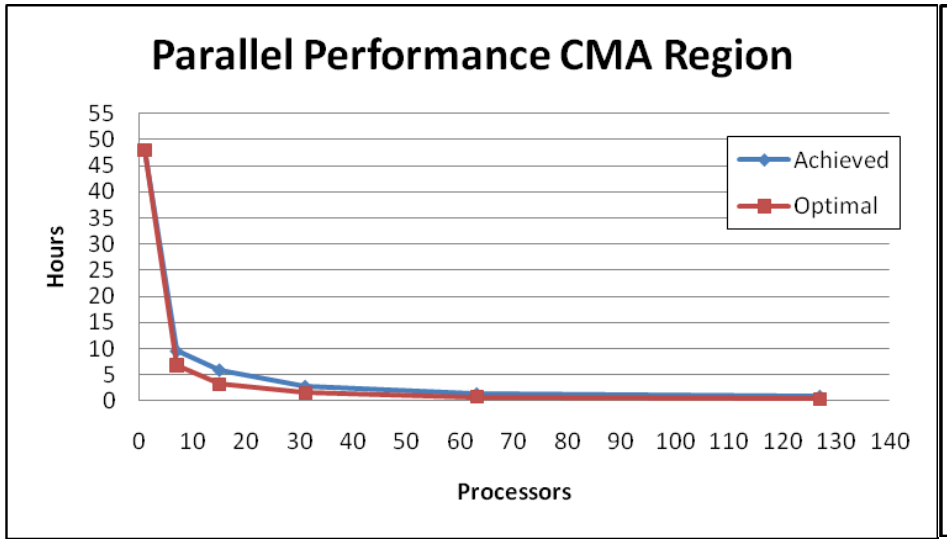


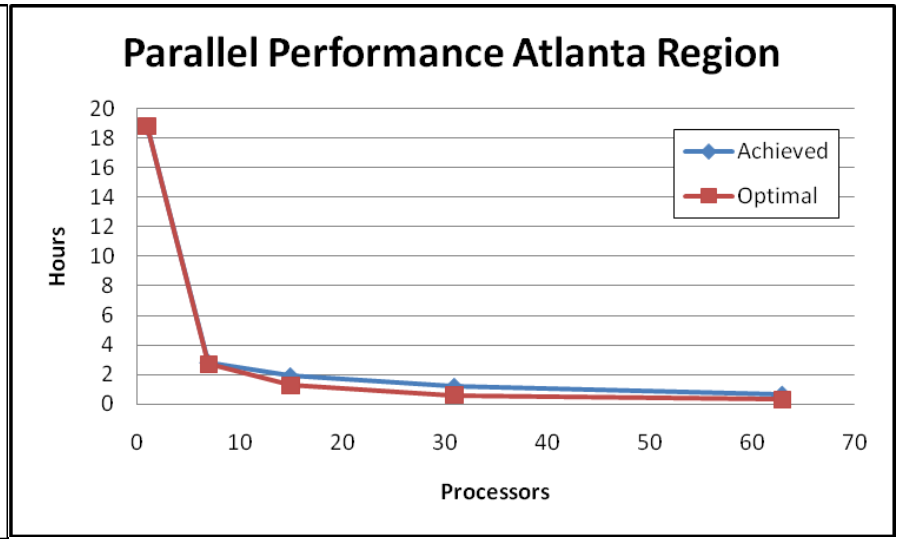
Atlanta Metropolitan Area

Regional Performance Results

CPUs (Slave)	Parallel (Hours)	Optimal (Hours)
1	48	48
7	9.633333	6.857143
15	6	3.2
31	2.85	1.548387
63	1.433333	0.761905
127	0.966667	0.377953

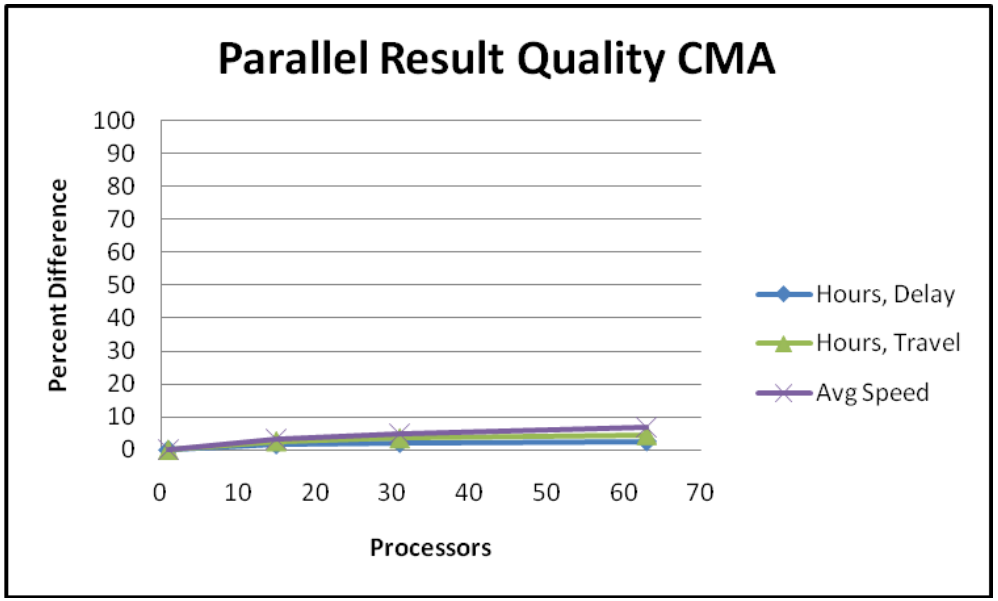
Processors	Parallel	Optimal
1	18.81667	18.81667
7	2.816667	2.688095
15	1.9	1.254444
31	1.183333	0.606989
63	0.65	0.298677





Result Quality Comparison

- Reasons For these discrepancies
  - Network Queue re-ordering across boundaries (cars become better ordered on links and actually perform better)
  - Various random number rolls (driver reaction time, permission probability, etc…)
  - Differing boundary link mechanics, not being able to look ahead produces different decisions



	1	15	31	63
Vehicle Hours of Delay	8928193	9083615	9124482	9165575
Percent Difference	0	1.725785	2.174632	2.623908
Vehicle Hours of Travel	11276526	11571852	11674123	11791863
Percent Difference	0	2.585093	3.464798	4.467899
Average Speed	12.69	13.11	13.33	13.6
Percent Difference	0	3.255814	4.919293	6.922784

Future Development

- The results indicate a stronger need for a partitioning weighting which will provide better initial load balancing.
- In addition there will be testing on a capability to dynamically balance the load by re-partitioning the simulation at certain points during the day.
- Parallel performance could still be improved by making several minor optimizations throughout the simulation code.
- A long term goal is to maintain parallelization through geo-partitioning across multiple nodes and to implement multi-threading for a single processor.