Accelerating Road Network Simulations using GPUs

Peter Heywood

The University of Sheffield

About Me

- MComp Computer Science & Artificial Intelligence at Sheffield (2010-2014)
- PhD Student at Sheffield (2014 - 2018)
- Research Software Engineer (RSE) and PhD Candidate at Sheffield (2018-2021)

Research
Software
Engineering
Sheffield.

Table of contents

Road Network Simulation

Road Network Simulation

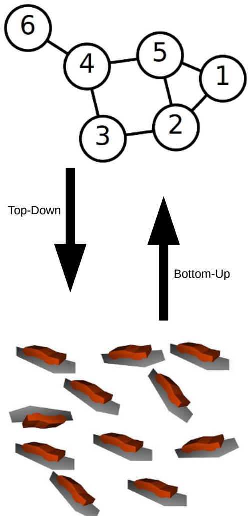
- Global transport demand is increasing
- Many constraints on transport networks
- Simulation can improve use of limited resources
  - Planning
  - Management

Road Network Simulation

- Simulations are becoming more computationally expensive
  - **Larger** - City-scale, National-scale
  - **More Complex** - CAVs, Smart Motorways, ...
  - **More Permutations**
- **Better-than Real-time** simulations required for active management
- Need more compute!

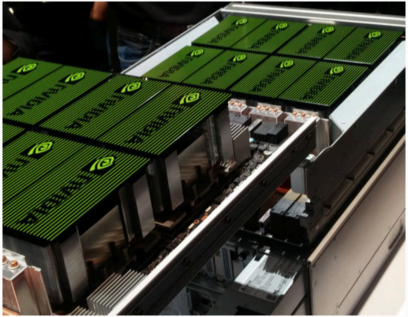
Simulation Categories

- **Macroscopic** Simulation (and Assignment)
  - Top-Down
  - High level, flow simulation
- **Mesoscopic** Simulation
  - Middle-out
  - Fine-grained Macrosimulation or Platoons/groups
- **Microscopic** Simulation
  - Bottom-Up
  - Low level, individual vehicles



Graphics Processing Units (GPUs)

- Massively Parallel co-processors
- Data-parallel algorithms and data structure
- Suitable for all scales of road network simulation
  - Different degrees of parallelism expressed
  - Different levels of performance improvement



NVIDIA DGX-2

GPU Accelerated *Macroscopic* Simulation

Macroscopic Simulation

- Top-Down Simulations
- Models networks as flows on roads (i.e pipes)
- High level of abstraction
- Relatively long time steps
  - Misses short-term events
- Low data requirements
- Lower computational cost

SATURN

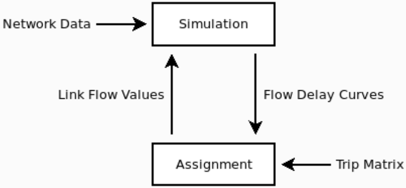
- Simulation and Assignment of Traffic to Urban Road Networks
- Commercial multi-core CPU software
- Originally Developed in the 1970s by Dirck Van Vliet at Leeds University
- Used by companies and governments for planning
  - Highways England
  - Transport for London (TfL)
  - Transport for the North (TftN)
  - etc.
- Fortran 77 with OpenMP

ATKINS
Member of the SNC-Lavalin Group

SATURN

SATURN Algorithms

- Iterative Equilibrium-based algorithm of Assignment and Simulation
  - Wardrop's Equilibrium
- **Assignment Phase**
  - Network + Demand Matrix -> Flow per road
  - Different vehicles types are considered independently (*User Classes*)
  - Trip Matrix contains many *Origins* and *Destinations*
    - Known as *Zones* or *Centroids*



Assignment-Simulation Loop in SATURN

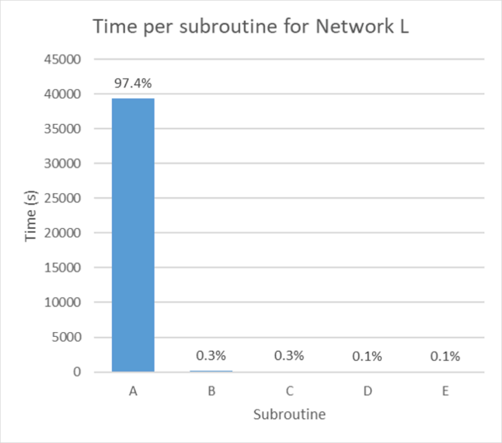
SATURN Models

- Range of scales from tiny to very very large
- Road networks are very sparse graphs
  - Preprocessing step to create a denser representation
  - Referred to as "*Spider Network*"
  - Contraction Hierarchies
- Network E is far too small for the GPU
  - Very useful for debugging!
- These are **Very Sparse** graphs, even when preprocessed

Model	Size	Centroids	Vertices	Edges
E	Town	12	17	74
D	Small City	547	2700	25385
C	Large City	2548	15179	132600
L	Metropolitan	5194	18427	192711

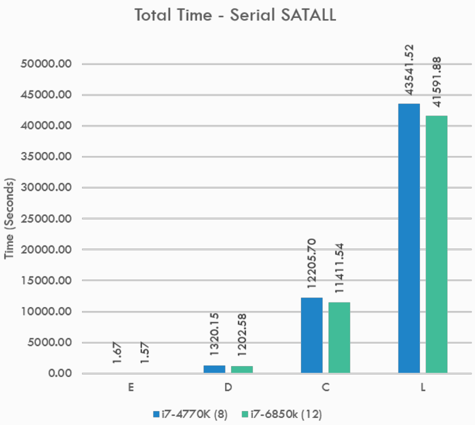
SATURN Profiling

- Serial version of SATALL
- Largest available model (L)
  - London + Surrounding area
- $>$ 11 Hour Runtime
- 97.5% in a single subroutine
- Candidate for Parallelisation
- Computes shortest paths, and traces them accumulating flow

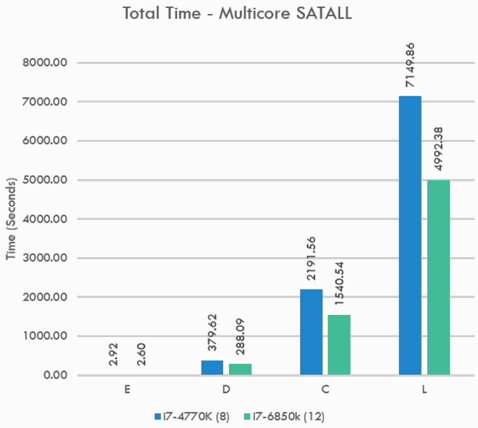


CPU Performance

Single Core CPU

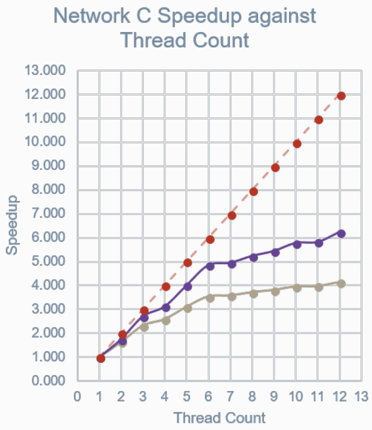


Multi-Core CPU

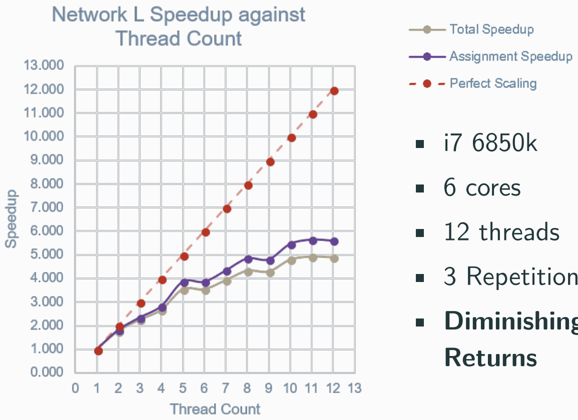


CPU Scaling

Single Core CPU



Multi-Core CPU



- i7 6850k
- 6 cores
- 12 threads
- 3 Repetitions
- **Diminishing Returns**

CPU Algorithm

```
For each User Class of vehicle
    For each origin centroid
        Calculate shortest paths (SSSP)
            For each destination centroid
                Trace the route updating flow (FA)
```

CPU Algorithm

- **Single Source Shortest Path (SSSP)**
  - Calculated for all origins
  - Typically $\frac{1}{4}$ of total nodes
    - All-Pair Shortest Path (APSP) algorithms would do too much work
  - Uses the D'Esopo-Pape algorithm
    - Algorithmic decision in the 1970s, due to benchmarking at the time
    - Switching to a modern implementation of Dijkstra's would likely yield a speed up
- **Flow Accumulation**
  - Trace routes between all origin-destination pairs
  - Update per-edge flow value at each step
  - Double precision to avoid numerical loss

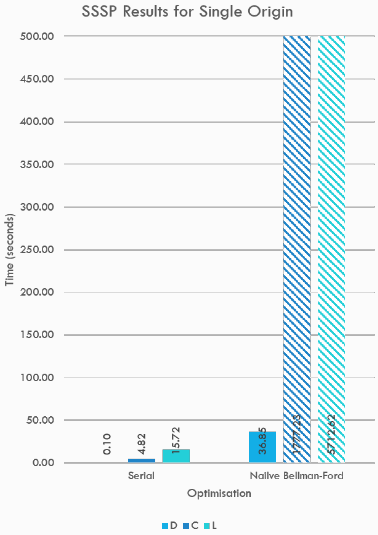
GPU Algorithm

```
For each User Class of vehicle (independent tasks)
    For each origin (centroid) calculate SSSP in parallel
    For each origin-destination pair accumulate flow in parallel
```

- Use the Bellman-Ford SSSP algorithm
- Highly Parallel, but much less-efficient than Dijkstra's or D'Esopo-Pape
  - For up to a worst-case number of iterations
  - Consider each edge in the network, updating routing information.

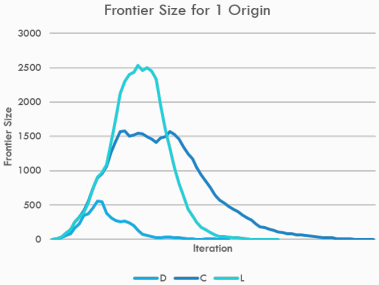
Initial GPU Implementation

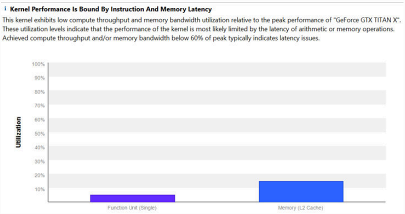
- Naive version of the Bellman-Ford Algorithm
- **Much, Much, Much, Much Slower…**
    - 364*x* slower
    - Inefficient use of compute
    - Inefficient transfer of data over PCI-e
- Non-deterministic
    - Different routes with the same cost
    - Order of execution is important
    - Still the correct result



Vertex Frontier

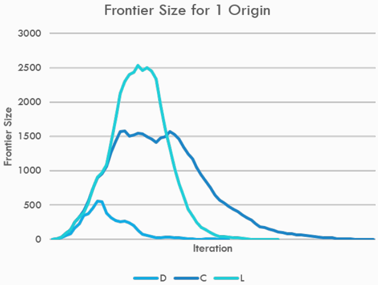
- Improve performance through algorithmic change
- Vertex Frontier tracks which vertices could result in an update
  - Increases Efficiency
  - Decrease Parallelism
  - Uses more memory
- Not enough Work
  - Latency bound
  - Low number of threads ($<$ 2500 for network L)

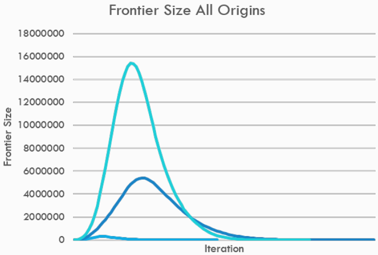




Origin-Vertex Frontier

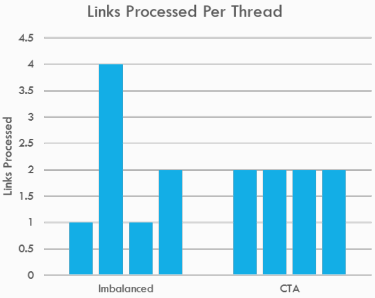
- Increase parallelism by solving multiple origins concurrently
- Origin-Vertex Frontier tracks which origins-vertex pairs could result in an update
  - Increases Parallelism
  - Uses much more memory
- Large amount of inactive threads
  - imbalanced work-load
- Poor data-access pattern
  - Lots of scattered accesses





Cooperative Thread Array

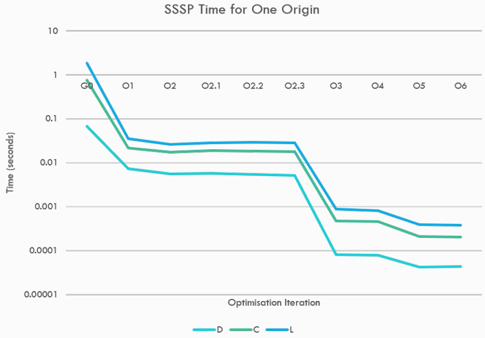
- Number of edges per node varies - imbalanced workload
  - Co-operative Thread Array (CTA)
  - Threads in a block collectively work on the same portion of the frontier
  - Balances work load
  - Improves L2 Bandwidth from 148GB/s to 716GB/s
  - CUDA 9.0 introduces clean methods to do this



L2 Cache			
Reads	233742509	490.399 GB/s	
Writes	107614703	225.779 GB/s	
Total	341357212	716.178 GB/s	Idle Low Medium High Max

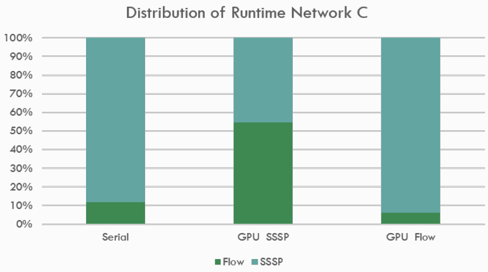
Iterative Improvements

- Profile and analyse performance after each change
- Implement possible solution, and profile again
- Resulted in
  - Changing data-layout to reduce atomic contention
  - Reduced memory usage
  - Improved Register Usage



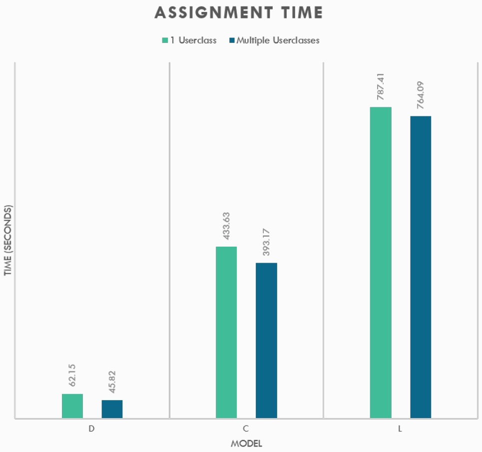
Change of Limiting Factor

- Flow Accumulation became the slowest part
- GPU Implementation using `atomicAdd` works well on modern hardware
  - `atomicAdd(double)` is a hardware instruction since Pascal
  - Software implementation on Kepler and Maxwell is very slow
  - Sorting based algorithm improves Maxwell performance, but still slower than Pascal



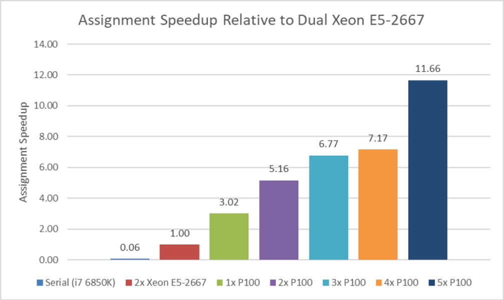
Multiple User Classes

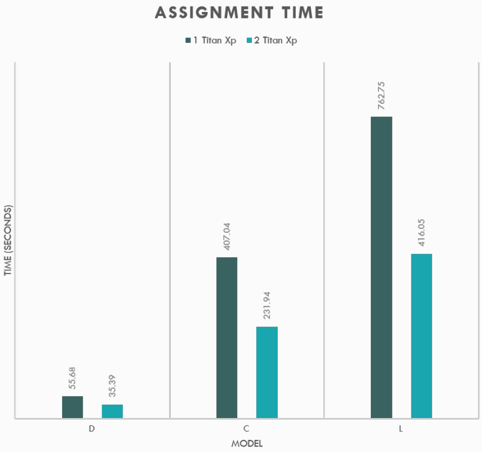
- User classes can processed independently
- CUDA Streams for concurrent processing
- Oversubscribes the GPU, allowing the device driver to load-balance SMs
- Provides more work to the GPU for small models
- Paves the way for multi-gpu



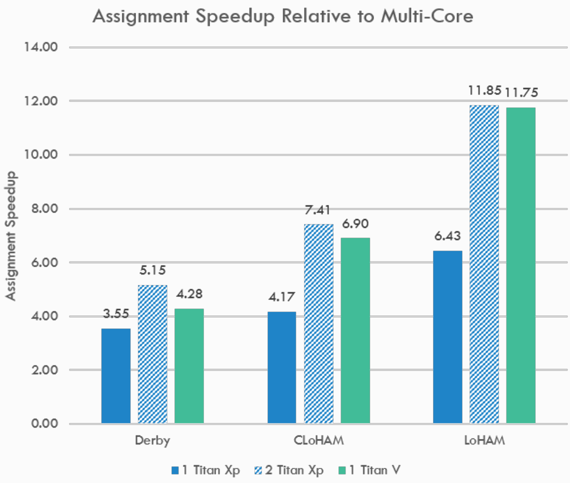
Multiple GPUs

- Independent user classes on each GPU
- Imbalanced workload between devices
  - Only assign whole user classes





Volta GPU Architecture

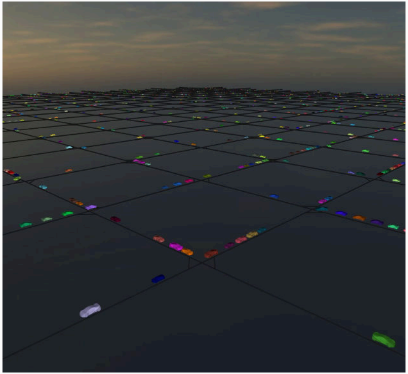


- Up to 80% performance improvement vs 1 Titan Xp
- Speed up relative to 6 core i7
- No source code changes
  - Other than updating libraries (CUB) and CUDA version.

GPU Accelerated *Microscopic* Simulation

Microscopic Simulation

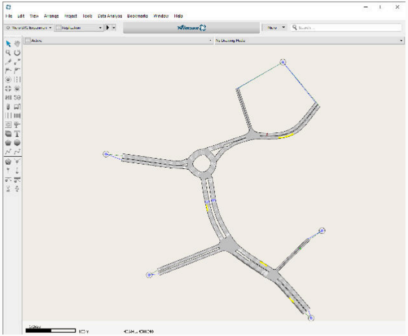
- Bottom-up Simulations
- Simulations individual vehicles and local interaction
  - with other vehicles
  - with the environment
- Agent Based Modelling (ABM)
  - Intuitive descriptions of behaviour and interactions
  - Complex behaviour emerges from simple rules
- Very Computationally expensive
- High data requirements



FLAME GPU Microscopic Simulation

Aimsun

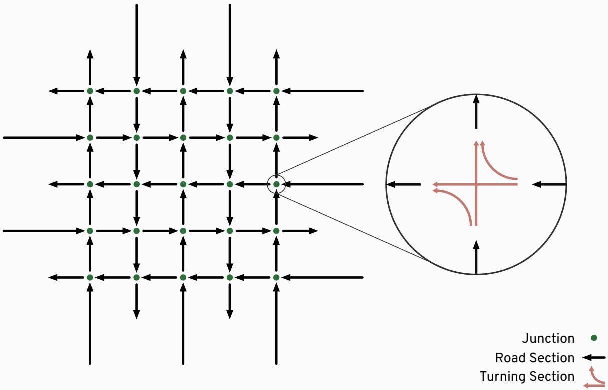
- **Aimsun**
  - Commercial multi-core CPU microscopic simulator
  - Used globally within the transport industry
  - Can simulate a broad array of transport networks and infrastructure



Aim

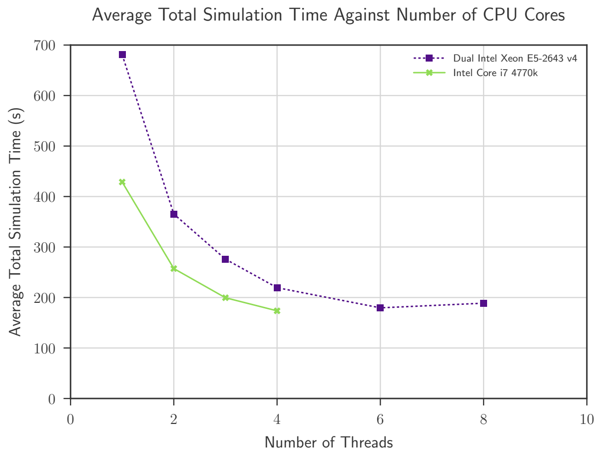
- Demonstrate GPUs are suitable
  - Implement a subset of models
  - Benchmark both applications on a scalable transport network

- Manhattan-style grid network
- Single lane, one-way roads
- Stop-signs at junctions
- Entrances and Exits at the edge of the simulated grid



Aimsun CPU Performance

- Single size of grid network
- 3 repetitions
- **Diminishing Returns** from additional cores



Models

- Gipps' Car Following Model
- Aimsun Gap Acceptance Model
- Turning Probability based Routing
- Simulated Vehicle Detectors
- Constant Vehicle Arrival

Gipps' Car Following Model

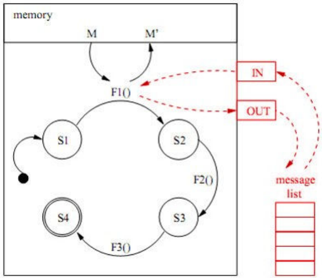
$$v_{free}(n, t+\tau) \leq v(n, t) + 2.5a(n)\tau(1 - v(n, t)/V(n))(0.025 + v(n, t)/V_t(n)^{\frac{1}{2}}$$

$$v_{safe}(n, t+\tau) \leq d(n)\tau + \sqrt{d(n)^2\tau^2 - d(n)(2[x(n-1, t) - s(n-1) - x(n, t)] - v(n, t)\tau - \frac{v(n-1, t)^2}{\hat{d}(n)})}$$

$$v(n, t+\tau) = \min\left\{v_{free}(n, t+\tau), v_{safe}(n, t+\tau)\right\} \tag{1}$$

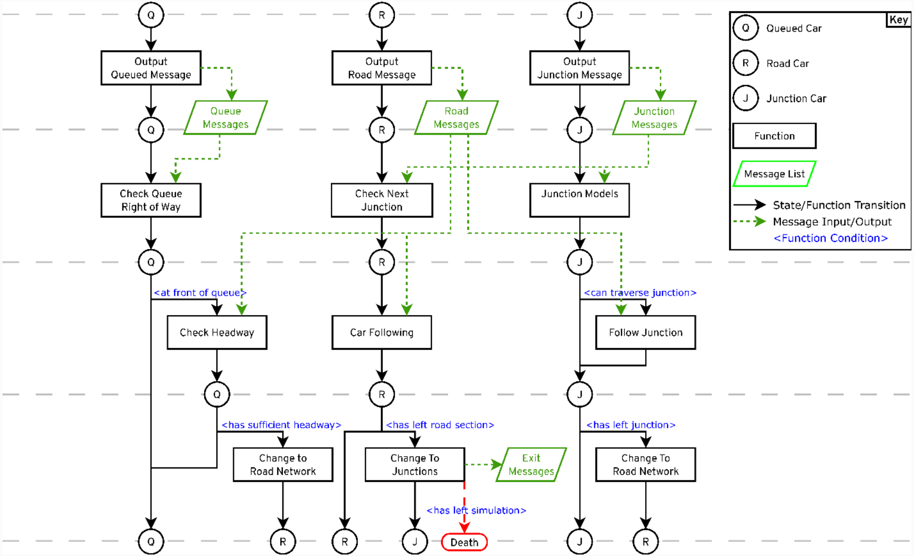
FLAME GPU

- Flexible Large-scale Agent Modelling Environment for the GPU
- Template-based simulation environment for high performance simulation
- Agents represented as X-Machines
- Message lists for communication
- High level interface for describing agents, abstracting the CUDA programming model away from the user.
- State-based representation minimises divergence and improves coalescence



flamegpu.com

FLAME GPU



Q
R
J
Output Queued Message
Output Road Message
Output Junction Message
Queue Messages
Road Messages
Junction Messages
Check Queue Right of Way
Check Next Junction
Junction Models
<at front of queue>
<can traverse junction>
Check Headway
Car Following
Follow Junction
<has sufficient headway>
<has left road section>
<has left junction>
Change to Road Network
Change To Junctions
Exit Messages
Change To Road Network
<has left simulation>
Death
Key
Queued Car
Road Car
Junction Car
Function
Message List
State/Function Transition
Message Input/Output
<Function Condition>

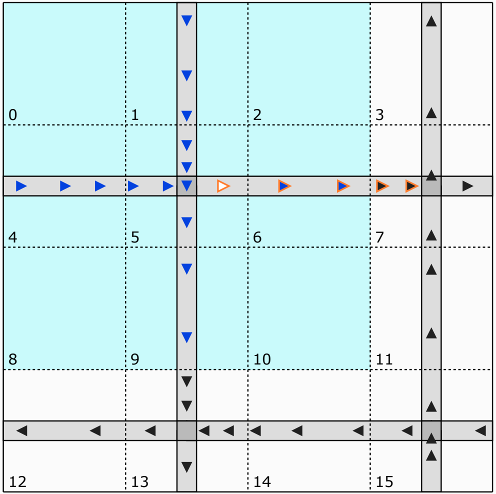
FLAME GPU Communication

- Message Lists enable high performance memory access pattern
  - and avoid issues with concurrent access to agent memory
- Typically the performance-limiting factor in FLAME GPU simulations
- Specialisation for typical communication patterns to improve performance
  - All-to-All
  - Discrete Partitioned Messaging (2D Cellular Automata)
  - Spatially Partitioned Messaging (2D & 3D Continuous Agents)
  - Non-optimal for road network models

On-Graph Communication

- Models typically need to access messages based on the transport network
- Couple messages to the graph
- Reduce the number of messages to be iterated by accessing messages from the relevant edge(s)
- I.e. Gipps' Car Following model only requires information from the lead vehicle

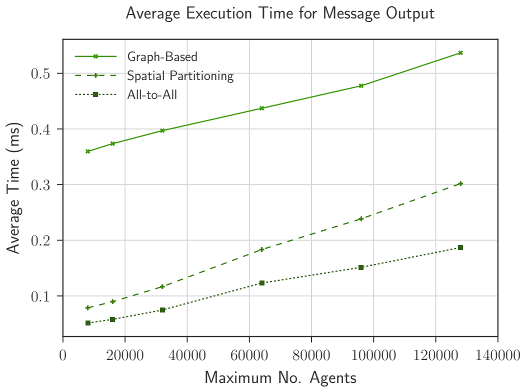
Communication	Messages
All-to-All	42
Spatial	18
Graph	5

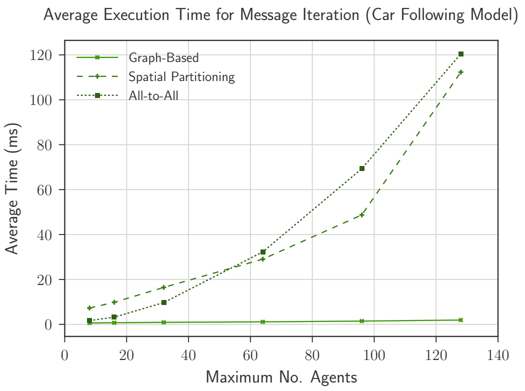


Example highlighting FLAME GPU Communication strategies

On-Graph Communication Performance

- Measured performance of Car following behaviour message output and input
- Higher output cost, **much** cheaper message input cost.





Performance Benchmarking

- Scale vehicle population and environment
- Scale vehicle population for fixed size environment
- 3 repetitions
- 1 hour of simulated time
- Multiple hardware configurations

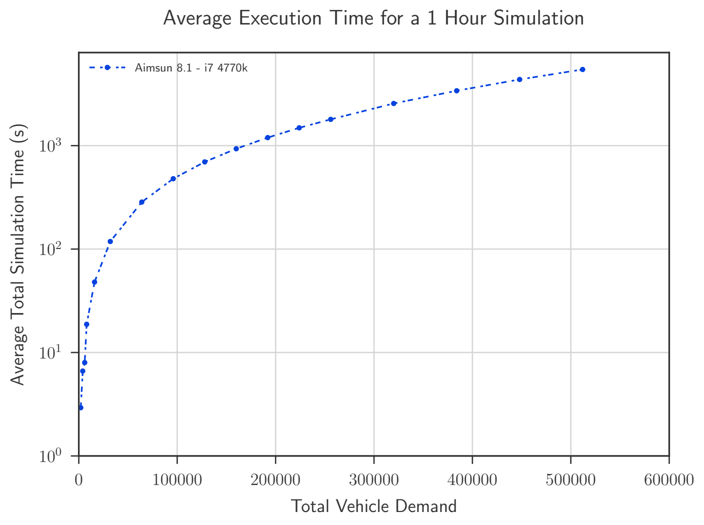
Workstation

- Windows and Linux
- i7 4770k (4 Cores)
- GTX 1080
- Titan X (Pascal)
- Titan V

Nvidia DGX-1

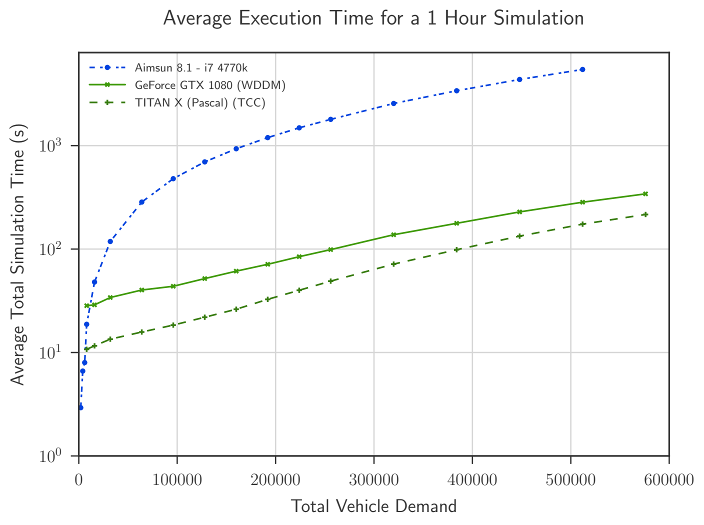
- Linux
- 2x Xeon E5 2698 v4 (20 cores ea)
- 8x Tesla P100

Population and Environment Scale



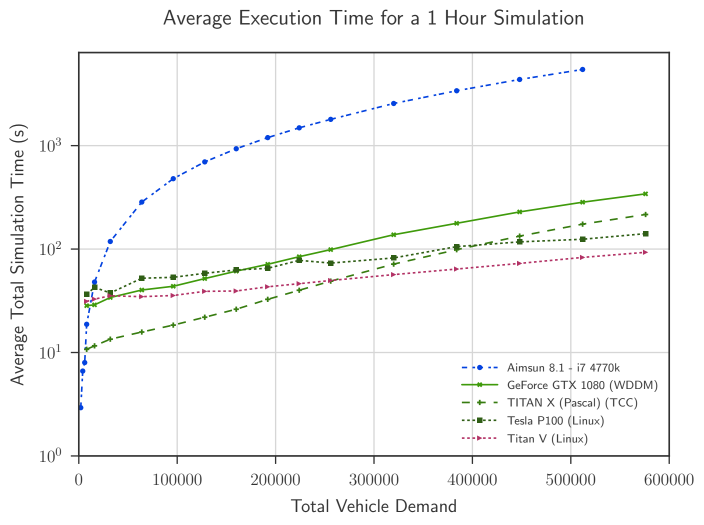
- 0.5 Million Vehicles:
- CPU - Windows
  - 5447s

Population and Environment Scale



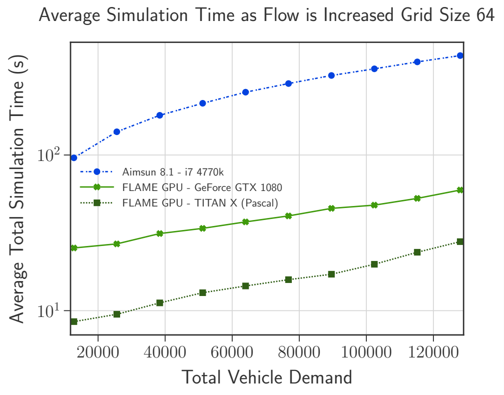
- 0.5 Million Vehicles:
- CPU - Windows
  - 5447s
- GPU - Windows
  - 174.2s
  - 31x speed up (Titan X (Pascal))

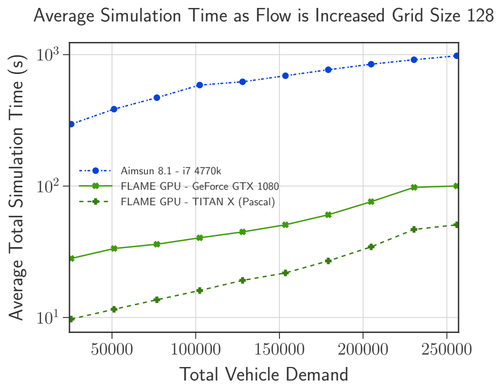
Population and Environment Scale



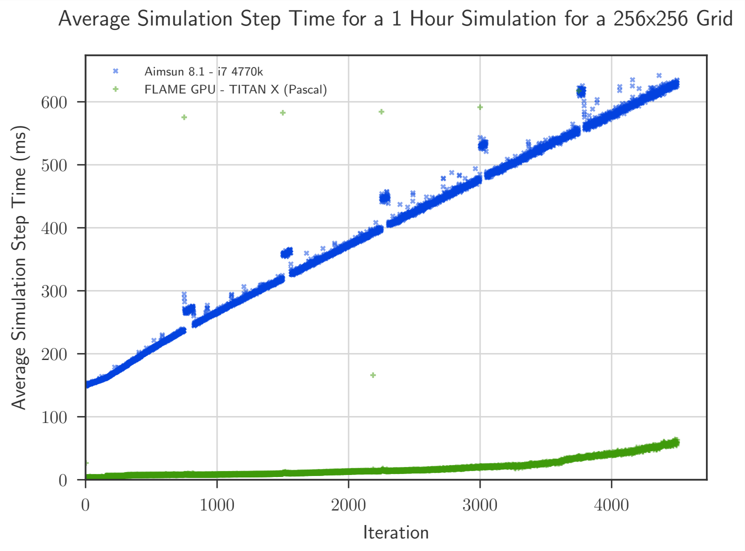
- 0.5 Million Vehicles:
- CPU - Windows
  - 5447s
- GPU - Windows
  - 174.2s
  - 31x speed up (Titan X (Pascal))
- GPU - Linux
  - 82.04s
  - 66x speed up (Titan V)

Population Scale for Fixed Environment





Runtime per Iteration

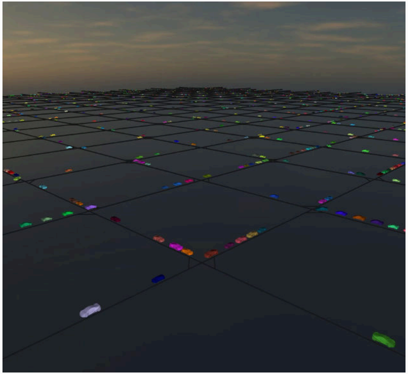


- Population grows as time progresses
- Anomalous values correlate with detector outputs
- Every 800 iterations (10 minutes)

Summary

Conclusion

- Macroscopic Assignment
  - Up to 11.7x speed up on 1 Titan V vs 6 core i7
  - Up to 11.8x speed up on 5 P100 vs dual socket Xeons
- Microscopic Simulation
  - Up to 66x speed up using a Titan V
  - Real-time-ratio of 39x for up to 576000 vehicles
- More simulations in less time
- Large simulations possible
- Better-than-real-time simulation of 0.5 million vehicles



Thank You

Supported By

- DfT Transport Technology Research Innovation Grant (T-TRIG July 2016)
- EPSRC fellowship "Accelerating Scientific Discovery with Accelerated Computing" (EP/N018869/1)
- Thanks to Atkins, STFC, TSC & Aimsun

Contact

- p.heywood@sheffield.ac.uk
- @ptheywood
- ptheywood.uk
- rse.shef.ac.uk

More Information

"Data-parallel agent-based microscopic road network simulation using graphics processing units"

https://doi.org/10.1016/j.simpat.2017.11.002