ACCELERATING TRAFFIC MODELS USING GPU-BASED TECHNOLOGY

Richard Bradley, Ian Wright, David Swain
Atkins
Roger Himlin
Highways England
Peter Heywood, Paul Richmond
University of Sheffield
Mark Mawson
The Hartree Centre, Science Technology Facilities Council
Graham Fletcher, Roland Guichard
Transport Systems Catapult

1. INTRODUCTION

Highways England is investing £10m in five new Regional Transport Models (RTMs) for which the SATURN macroscopic traffic modelling software has been chosen. The RTMs scale means that model runtimes are up to five days per test and increasing model speed would greatly benefit the delivery of the RTMs and subsequent analysis.

In recent years modelling software has started to embrace parallelisation techniques with the introduction of CPU multi-core threading, and SATURN has already implemented CPU multi-core parallel computing. However, whilst each CPU thread has significant 'clock' speed they are limited in number with a typical modelling PC having a maximum of 20 threads. With the concept of parallel computing within assignment models already established, Highways England has now turned to GPU-based technology as a potential way of creating a step-change reduction in traffic model runtimes. GPU parallel computing has a number of differences to the equivalent CPU computing and, with a typical modelling mid-range PC, GPUs provides upwards of 8 TFLOPS (Terra FLoating point Operations Per Second) of theoretical performance in contrast with roughly 0.4 TFLOPS available in modern Intel Core i7-based processors.

Preliminary investigations into serial and CPU multi-core execution have highlighted that the areas of the highway modelling process that currently take significant computer resources, in terms of memory and compute time, are associated with the assignment of traffic to the model network, and specifically focussed on tree building algorithms. The processes used in SATURN, or more specifically the SATALL assignment 'engine', are typical of other macroscopic traffic modelling software and therefore **any solution identified can potentially be applied more widely across the modelling industry.**

To investigate SATALL GPU implementations Highways England and Atkins Limited ('Atkins') have co-funded the first major transport modelling collaboration between public and private sectors, bringing in support from Transport Systems Catapult (TSC), University of Sheffield (NVIDIA partner) and The Hartree Centre (high performance computing). The team was challenged to find a step change reduction in runtimes in the first transport modelling application using GPU-based technology within the study's six-month timeframe.

2. WHY USE GPU-BASED TECHNOLOGY?

2.1 CPU and GPU-based technology

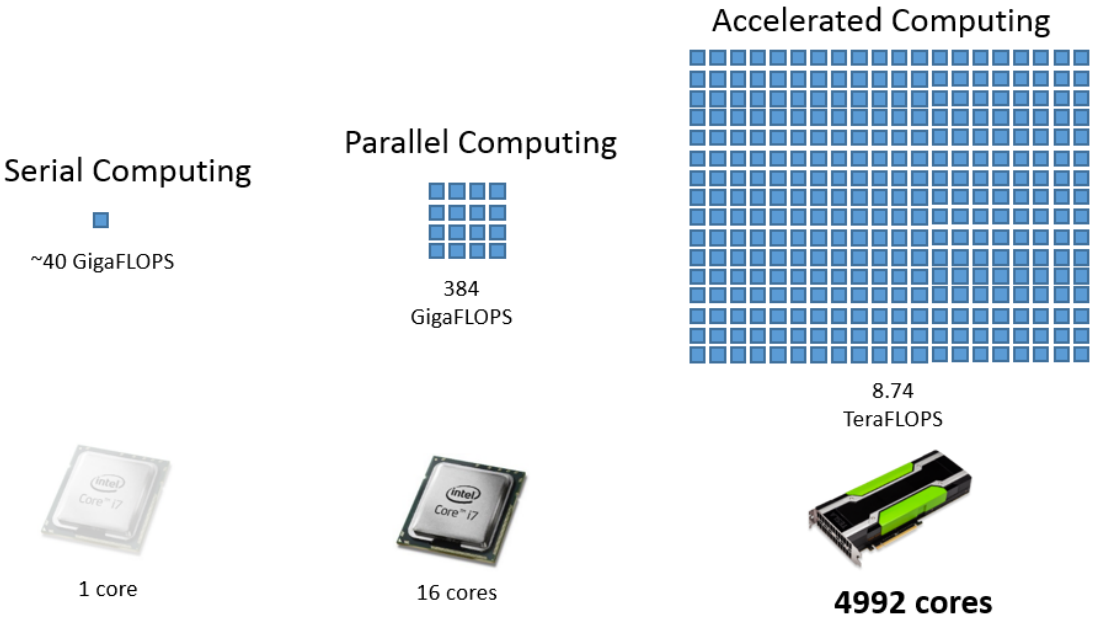
Computing hardware has been increasing in speed at a rate largely represented by Moore's law since 1965. However, it has slowed in recent years (reflecting the increasing difficulty in making transistors smaller) but reducing transport model runtimes remains a high priority due to the increasing number and size of large transport models enabled by generic, maintained datasets. This continuous need is reflected in the Moore's law compensator known as Wirth's law which is the principle that successive generations of computer software increase in size and complexity, thereby offsetting the performance gains predicted by Moore's law.

To help continue Moore's law a new definition of the law with greater collaboration between hardware and software may be necessary. As the growth in CPU speed is no longer expected to continue at the same historical rate, re-thinking algorithms to be massively parallelised could allow highly scalable data-parallel computing to provide massive economies of scale. This is unlikely to be achieved with CPU-based hardware as CPUs were primarily designed to execute serial code and extract maximum parallelism out of serial execution to improve performance. GPUs on the other hand are purpose built parallel computers which are fed parallel workloads. In fact the entire real-time graphics environment is designed around the massively parallel nature of GPUs. Another relevant difference between CPU and GPU is cost. GPUs typically offer greater performance figures for lower cost and also lower power usage per FLOP.

The approach to achieving continued step-change reduction in model runtimes is illustrated as three phases in computing technology **Figure 1** below. The step change from CPU 'serial' to CPU 'parallel' computing is now well-embedded within the transport modelling software, including SATURN

Multi-Core for example. This has generally involved relatively minor changes to the software to spread the workload over a small number of limited but fast cores operating in parallel. Typically this might include spreading the workload over 12 cores, and retains the algorithms designed to optimise serial performance whilst minimising duplication in the parallelised code.

Figure 1 Step Changes in Computing

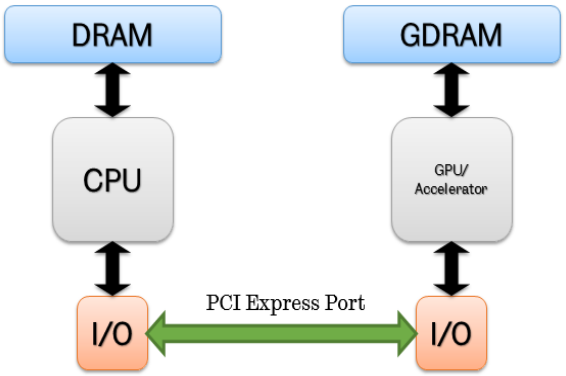


The step change from CPU parallel to GPU 'accelerated' computing is enabled through an even greater collaboration between hardware and software. Accelerated computing improves the execution of a specific algorithm by allowing greater concurrency, and reducing the overheads associated with managing this greater concurrency. As such the algorithms used must be suitable for simpler more repetitive calculations, with fewer iterative operations and less regard to wasted effort.

In an accelerated system both the CPU and GPU play important roles. The CPU, and general computing hardware, handles the operating system and provides the interface between the software, and the user and data, as well as managing the generic processes within the program. Unlike in parallel computing, where the CPU will also handle the more demanding computations within the software, the nature of accelerated computing is to package-up the massively parallelisable computations and send these to the GPU, thus running independently of the CPU and general computing hardware, as shown in **Figure 2** below.

As shown in the figure data has to be transferred to the GPU via the PCI express port. Managing this transfer of data becomes an important aspect of the achievable speed in accelerated computing, due to the high memory latency of the PCI express port, and therefore a key area for investigation.

Figure 2 Relationship between CPU and GPU



The data transferred to the GPU needs to 'parallel' data so as to allow the tasks to be performed on the data to also be parallel. It is therefore also important to ensure the data and the algorithm are consistently highly parallelisable throughout these key processes.

2.2 Algorithmic changes

It was also important to understand at an early stage the specific areas / routines in the SATALL code that were the most computational intensive. This was achieved by 'profiling' existing runs of the SATALL assignment / simulation. This profiling confirmed that the bulk of the runtime was associated with the building of 'paths', with paths representing a series of nodes through the network to route from one network node to another. It was possible to track the majority of the runtime to the path build algorithm, which uses the d'Esopo-Pape algorithm (which is not unusual for macro assignment models). This confirmed that if the assignment algorithm, and the data fed into the algorithm, could be massively parallelised then a step-change in runtime may be achievable. However, and as defined in Amdahl's law, any serial aspects of the code will be the ultimate runtime constraint.

A key decision for the algorithm is very much dependent on the choice of CPU or GPU due to the type of parallelism possible. The first parallelism option is 'task parallelism' where different threads are used for different tasks, and this is best suited for CPU parallelism. For example, if a system is running code with thread A and thread B, and we wish to do tasks X and Y, it is possible to tell thread A to do task X and thread B to do task Y simultaneously.

The second option is 'data parallelism', which is best suited for GPUs and focuses on distributing the data across parallel computing and is achieved

when each thread performs the same task on different pieces of distributed data. For example, thread A and thread B can undertake the same task X on data D. It is possible to tell thread A to do task X on one part of data D and thread B on another part simultaneously. Consider adding two matrices where Thread A could add all elements from the top half of the matrices, while thread B could add all elements from the bottom half. Since the two threads processors work in parallel, the job of performing matrix addition would take one half the time of performing the same operation in serial using one thread alone. If the matrix was say 5,000 zones, and there were 5,000 threads available, then each cell could be added simultaneously.

Within the existing SATALL path-building routine the majority of serial execution is based on the Single Source Shortest Path (SSSP) computation, which can be done either using task parallelism using the existing d'Esopo-Pape (or an alternative like Dijkstra), or in data parallelism, and using an alternative algorithm that is not focused on optimising serial processes.

2.3 Hardware considerations

Knowledge of likely hardware to be used can affect choices in optimising the code, via settings in the compiler, plus optimisation of the transfer of data across the PCI express port. It was also considered important to identify a specific hardware setup that would be typically used by the RTM modelling consortia. So, whilst a number of different graphics cards were tested, the main optimisation and testing was undertaken using the CUDA compute capability 5.0, optimised for the NVIDIA Titan X (Maxwell) GPU.

The existing SATALL code uses Double Precision (DP) to accumulate link flow totals. It was important to retain this precision to ensure stable model runs. However, Maxwell GPU hardware such as the Titan X only offer 1/32 DP performance compared to Single Precision (SP). Whilst alternative cards / architectures can provide better DP performance, they were not within the agreed target price range for typical model users. Accordingly, the project continued with this target hardware for investigating speed enhancements, and retaining flow accumulation accuracy became an additional area requiring investigation.

3. EXISTING SATURN SOFTWARE

3.1 Model structures

The SATURN congested assignment model is a complex procedure in which successive attempts to find the quickest routes across the network for each journey (the **SATASS** stage) are interspersed with updating calculations about the conflicting volumes at each junction and the nature of the delays that are likely to be faced by each junction user (the **SATSIM** stage). The overall process is illustrated in **Figure 3**.

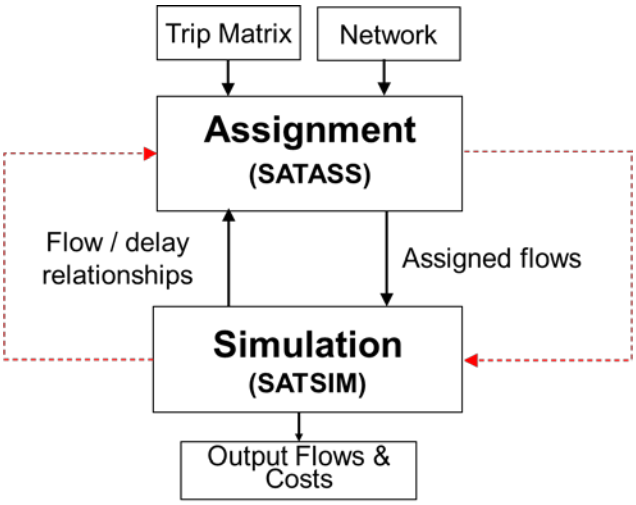


Figure 3 SATURN assignment structure

After several loops of this SATASS / SATSIM process, an equilibrium position is achieved in which traffic flows and junction delays are relatively stable in successive loops. At this point, the SATASS / SATSIM process is deemed to have converged, little can be gained by carrying out more loops of the program and the results can be regarded as sufficiently robust.

The SATURN Multi-Core add-on focuses on the SATASS only and undertakes the path building and loading as multi-threaded process for each origin. The GPU-based technology focusses on the same processes.

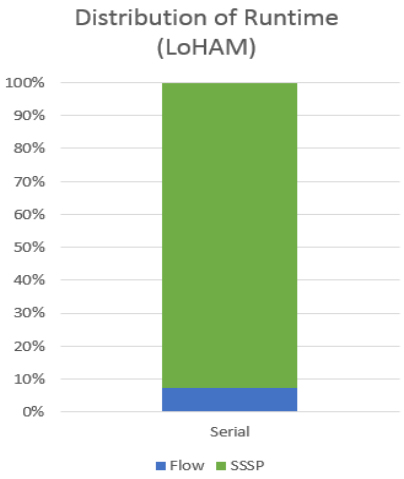
SATURN networks are coded as: 'buffer' to represent node / link topography and link based capacity restraint; and 'simulation' to represent junction capacity restraint. SATURN includes an internal network aggregation called 'spider' links that use network aggregation where links and / or nodes in the basic assignment network are combined together into an equivalent set of aggregated links / nodes with the objective of reducing the runtime required to carry out the basic assignment steps of path building.

The three distinct steps in SATALL assignment include: route choice; accumulate flow; and cost skimming. The route choice includes building SSSP, using an 'all-or-nothing' path build for each i-j pair. The flow accumulation then applies the T_{ij} trips to each link along the path. The cost skim then accumulates the costs along a path associated with the flows on each link.

Figure 4 shows the runtime profile of the SSSP and flow accumulation (referenced 'Flow') in the context of running the London Highway Assignment Model (LoHAM) in serial execution.

The route choice and building of paths was identified as taking 97.4% of the LoHAM runtime in serial execution, which was concentrated on calls to the SSSP algorithm. The profiling showed that the three routines of route choice, flow accumulation and cost skimming represented the highest computational demand and all three included the tree-build algorithm. Within this algorithm, the highest demand was for the routine called Load-It, which was therefore investigated first.

Figure 4 SATALL Assignment



The SSSP algorithm finds the node based path for each single origin to each network node with the lowest cumulative travel cost. The SATURN SSSP algorithm is based on d'Esopo-Pape algorithm. There are numerous other SSSP algorithms, including the popular Dijkstra, but d'Esopo-Pape was considered the best algorithm during the original SATURN development (and, with various optimisations, continues to perform very efficiently).

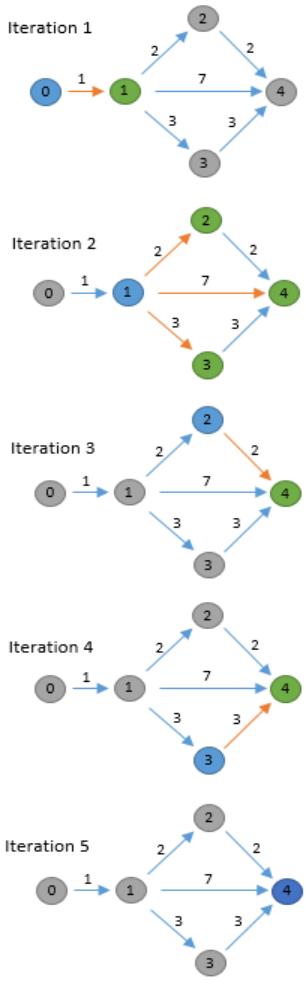


Figure 5 SSSP algorithm

Core to understanding the d'Esopo-Pape algorithm is that it maintains a 'priority queue' of nodes to explore shortest paths, which is common to all SSSP algorithms that are efficient when executed in serial. The algorithm starts by initialising the cost back from each node to the origin to infinity with the exception of the origin, which is initialised to 0.

In the example in **Figure 5** the cost of getting to origin 0 is set to infinity for nodes 1 to 4. In iteration 1 the algorithm looks for connecting nodes, finds node 1 and updates the cost of access origin 0 from node 1 to the associated link cost of 1 unit. Node 1 is then added to the 'loose end' array,

containing all nodes but prioritised by the cheapest cost first, which at this iteration will have node 1 first. In iteration 2, the algorithm starts at the top priority loose end node of 1 and accumulates the costs from connecting nodes 2, 3 and 4. These nodes are then added to the loose end array, with node 2 prioritised first with the shortest path of getting back to origin 0 of 1 + 2 = 3 units. The algorithm searches through the loose end table nodes in this way until all nodes have been visited, identified as the node cost back to origin not being infinity, with the shortest path stored in the 'back node' array.

This is a highly serial process because the priority queue is processed in order. As such d'Esopo-Pape is not massively parallelisable, and in particular data parallelisable, as required for GPU-based technology.

The flow accumulation is, however, a parallel problem as each path from destinations back to a single origin can be processed concurrently. However, when the link flow is accumulated this needs to be synchronised with the other concurrent paths, which presents additional computational speed restrictions.

3.2 Test Models

A number of real-life SATURN traffic models were required for testing purposes, representing a range of sizes to allow quick coding checks, and to provide evidence that the changes are likely to work for the RTMs. As such four models were used as shown in **Table 1** below, with the Central London HAM (CLoHAM) closest in size to the proposed RTMs.

Table 1 Test model summary

Model	Size	Zones	User Classes	Simulated Junctions	Typical Runtimes (SATASS & SATSIM)	
					Serial	Multi-Core
Epsom	Test	12	2	17	1.6 seconds	1.4 seconds
Derby	Medium	547	13	3,686	0.47 hours	0.14 hours
CLoHAM	Large	2,548	5	12,932	4.18 hours	0.62 hours
LoHAM	Extra Large	5,194	5	25,575	15.4 hours	1.80 hours

Hardware: Intel Core i7 2nd Gen Desktop (Four Core + HT)

4. SOFTWARE DEVELOPMENT AND INTEGRATION

4.1 Algorithm change

The existing use of the d'Esopo-Pape algorithm, or indeed any efficient serial SSSP approach, needed to be changed as it is the focus for runtime demand,

and the scalable and step-change in runtime requires data parallelism to allow the best use of GPU-based technology.

A literature review considered a number of SSSP algorithms that indicated a speedup potential in serial and alternative algorithms that could be implemented efficiently in a data parallel approach. The literature review found a 2014 paper titled 'Work Efficient Parallel GPU Methods for SSSP' detailed efficient data parallel methods. The methods discussed were implemented using the Gunrock library, developed by academics at the University of California, and which is relevant as it demonstrates the techniques could be implemented in a transport model. A range of GPU accelerated node-link network processing algorithms were explored. The paper showed that GPU accelerated methods out perform a heavily optimised Dijkstra implementation on a number of different networks. Furthermore the Gunrock library promotes a high performance implementation of the Bellman-Ford algorithm, which became the focus for this project.

The Bellman-Ford algorithm iterates incrementally from 1, each time incrementing the number of nodes found in the shortest paths, and up to the total number nodes minus 1, which is the theoretical maximum number of times before the longest shortest-path has been found. The key difference is that Bellman-Ford does not maintain a prioritised loose end array, as with the d'Esopo-Pape algorithm. Instead each link in the network is considered at each iteration, ensuring that all connected shortest paths are found.

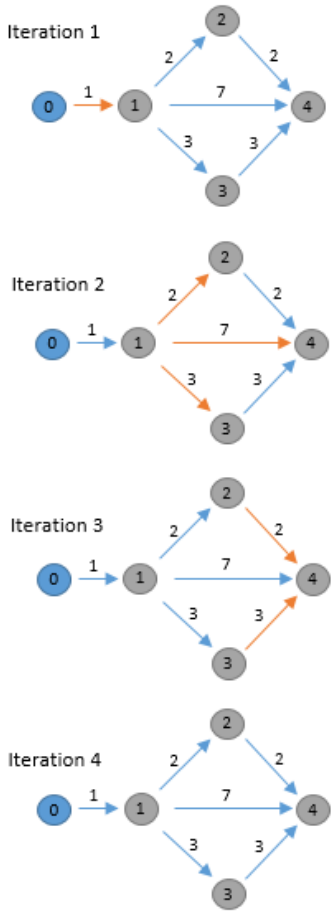


Figure 6 Parallel SSSP Algorithm

Both d'Esopo-Pape and Bellman-Ford algorithms produce the same output, which includes an array of back nodes and an array of back node costs, allowing a similar interface. **Figure 6** shows the iterative algorithm on small example network. For the origin node 0, all links are considered at each iteration, but only links coloured orange result in an update of the back node arrays, as it looks for the shortest path with the number of nodes indicated by the iteration

number. Effectively iteration 4 is redundant but that is not known until it has been processed.

As the Bellman-Ford algorithm evaluates all links in the network at each iteration, it is a poor candidate for serial or task-parallel execution as more efficient serial and task-parallel algorithms exist. In the example Bellman-Ford considers 6 links at each iteration (24 links in total) to find the shortest paths, whereas D'Esopo-Pape only considers 6 links through the efficient use of a priority queue array. However, Bellman-Ford is an ideal candidate for data-parallel execution as each link can be considered simultaneously at each iteration. Furthermore, computing the shortest paths for multiple origin zones concurrently allows greater levels of parallelism to be exposed, making better use of the highly-parallel GPU hardware.

4.2 Algorithm route choice development iterations

The algorithm development was undertaken using data extracted from SATALL at the point of beginning a SSSP routine and the corresponding output, and the SSSP code extracted as the source for the first algorithm. A simple harness would provide this data to the algorithmic code under test, without requiring recourse to SATALL. This enabled a rapid turnaround in tests to enable early development of the new algorithm while integration and compiler issues were resolved. An iterative agile process was followed to update the extracted SATALL software to introduce, and then optimise, the Bellman-Ford algorithm, with the first stage associated with the extracted Load-It route choice code. The list below details each iterative change of the extracted Load-It code and **Figure 7** below shows the speedup achieved.

Iteration A1 – D'Esopo-Pape Serial
D'Esopo-Pape algorithm in serial execution used for reference.

Iteration A2 – Naïve Bellman-Ford
Simplest possible implementation of Bellman-Ford with no optimisation. As anticipated this resulted in a poor performance, to such an extent that it went off the scale in the runtime figure, as indicated by the striped bars.

Iteration A3 – Bellman-Ford Early Termination
Bellman-Ford will continue to iterate to a maximum number of iterations based on the number of nodes in the network minus 1. However, it is not always necessary to undertake all iterations and therefore the code terminates if two

successive iterations have the same costs. This provided good runtime improvements over the A2.

Iteration A4 – Bellman-Ford Node Frontier
Introduction of a node frontier - an array which contains the nodes updated in the previous iteration. Only links which leave nodes in the frontier array are considered, saving a considerable amount of work. Modest improvement over A3 but significantly slower than the reference runtimes from A1.

Iteration A5 – Bellman-Ford Load Balancing
Different nodes have different numbers of connected links. Instead of the links from a single node being processed by a single thread, a group of threads co-operate to process the links from the group of nodes, balancing the workload. This change introduced little improvement over A4.

Iteration A6 – Bellman-Ford All Nodes
The existing D'Esopo-Pape and A5 Bellman-Ford algorithms stop at the node linking to the destination zone link connector, and then add the zone connector in serial after the path build. This change moves the destination zone into the process so it can be parallelised. This change introduced little improvement over A5, but increases the amount of serial code replaced.

Iteration A7 - Bellman-Ford Larger Results Arrays for Multiple Origins
The back node and back node cost arrays were increased in size to store the results for multiple origins. This enables multiple concurrent origins and also improved memory access. This change introduced little improvement over A6.

Iteration A8 – Bellman-Ford Multiple Origins
Parallelise and find the shortest paths for all origin zones concurrently. This provided the desired step-change in runtimes compared to the reference runtimes from A1.

Iteration A9 – Bellman-Ford Dynamic Batching
Origin batching was added to dynamically configure the data sent to the GPU to improve the use of memory and allow execution of models too large for a single GPU. This change introduced little improvement over A8.

Iteration A10 – Bellman-Ford Improved Load Balancing
Implemented a different search strategy to better optimise the load balance across threads. This change introduced some modest improvements over A9.

Iteration A11 – Bellman-Ford Interleafed Back Node & Cost Array

The back node and back node cost arrays were 'interleafed' to improve memory access patterns and reduce the number of atomic operations which add significantly to the runtimes. Atomic operations are operations which are guaranteed to occur in order (serialised) and be required to avoid race conditions. Interleaved results allow a reduction in the number of atomic operations required and therefore a reduction in serialisation. This change introduced little improvement over A10.

Other points to note following the improvements, but that were omitted from the Load-It code changes, are listed below.

- GPU acceleration for non-spider networks was not implemented due to the complexities of 'u-turns' on the buffer-simulation network boundaries having significant performance impact on Bellman-Ford. As most networks will always use spider networks in the path build (for speed reasons and the default option) this was considered acceptable.

- Memory must be initialised for each path build; asynchronous memory initialisation for large arrays typically makes full use of the GPU, leading to serial execution and a minor reduction in performance due to additional overhead costs.

- Alternate back node storage with the interleaved back node array stored by origin and then node, i.e. |o1n1|o1n2|...|o2n1|o2n2|... could instead it could be stored |o1n1|o2n1|...|o1n2|o2n2|.... However, this did not improve overall performance but if the A4 frontier version was also modified to be interleaved it may result in a net increase.

- Batching sets of origins for SSSP calculation would allow concurrent

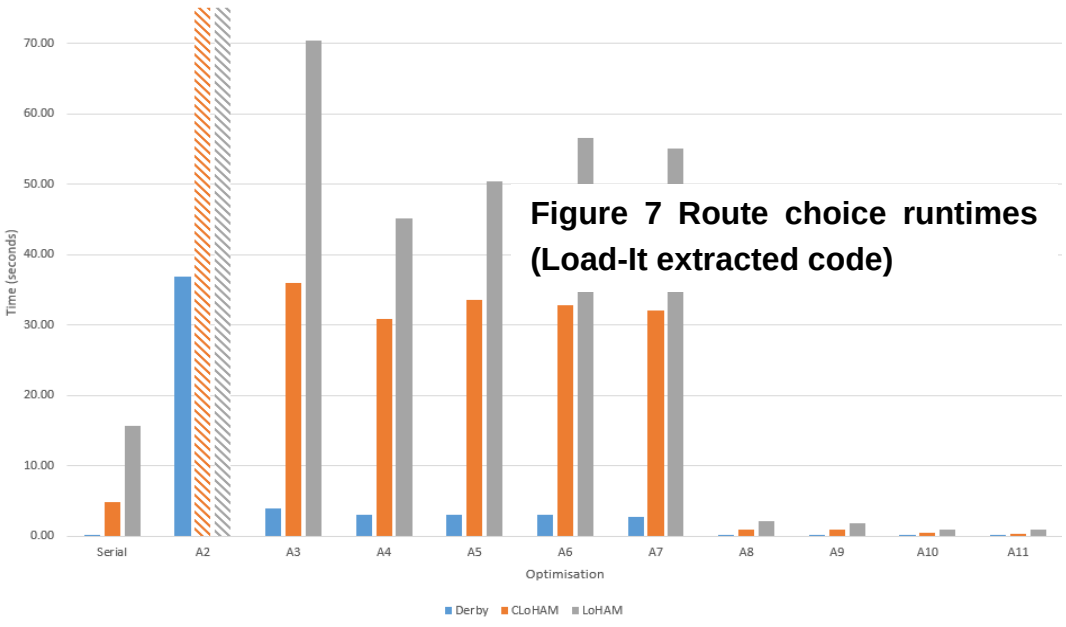


Figure 7 Route choice runtimes (Load-It extracted code)

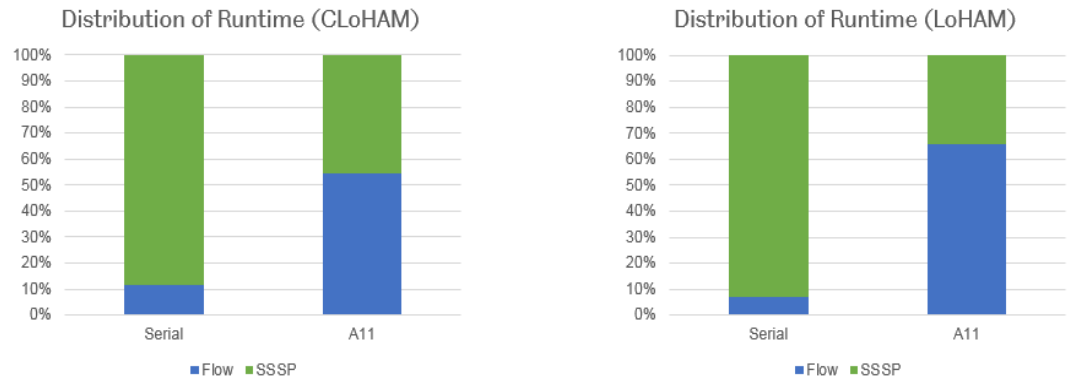
batch execution allowing the GPU driver to better manage hardware availability, however doing so results in a greater number of expensive device-host data transfer over the PCI express port. The cost of the additional data transfer outweighs the benefits.

- Dividing the main SSSP function into smaller units (as implemented in the Gunrock framework) would enable improved memory access patterns. This however results in a much larger memory footprint reducing the number of concurrent origins being processed on the GPU simultaneously. Early investigation led to a loss of performance due to the reduced level of parallelism.

4.3 Algorithm flow accumulation development iterations

Towards the last iterations of the optimisation of the Bellman-Ford route choice implementation the SATALL profile indicated that the runtime focus was moving from the route choice to the flow accumulation. **Figure 8** shows that for iteration A11 the flow accumulation represented the majority of the runtimes in the full SATALL code, and therefore this became the next focus.

Figure 8 Distribution of runtime following A11 Bellman-Ford route choice implementation



The flow accumulation process uses the shortest paths code produced by the Bellman-Ford algorithm from A11 with the origin-destination trip matrix to calculate the flow per link in the network for the current user class. For each trip in the matrix, the route is traced from destination back to the origin, accumulating the flow value as each link is encountered. This process is performed on the GPU, with the challenge of providing a speedup whilst producing the results with the correct precision.

Unfortunately the GPU hardware (Maxwell) used during this project does not support a key atomic operation in hardware for DP floating point numbers. Two alternative solutions were investigated: firstly the use of SP for the accumulation of flow was investigated along with options for minimising the loss of precision; and secondly the use of parallel primitives to minimise expensive software-based atomic operations was investigated.

Again an iterative agile process was followed to update the SATALL software to introduce, and then optimise, the Bellman-Ford algorithm, with the second stage associated with the flow accumulation to the array 'Flow8'. The list below details each iterative change using the now integrated Load-It routine, of which the flow accumulation is part, and **Figure 9** shows the speedup achieved.

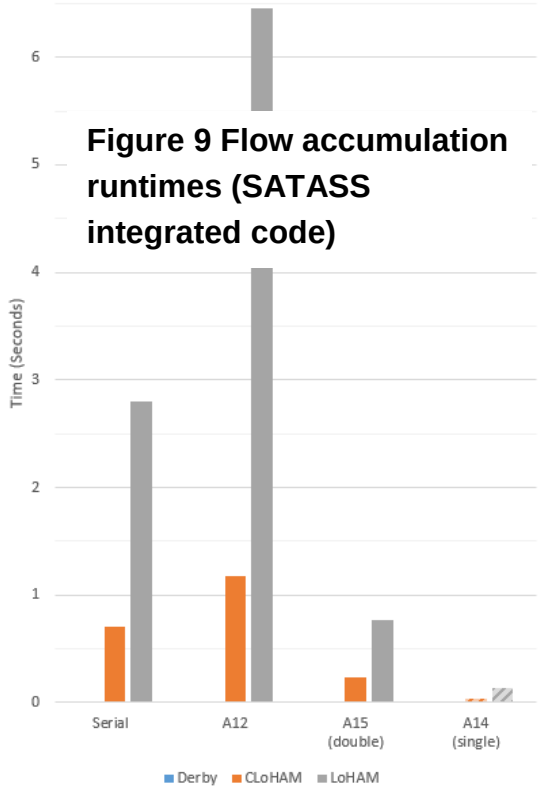


Figure 9 Flow accumulation runtimes (SATASS integrated code)

Iteration A12 – Naïve DP Flow8
Retains Flow8 DP on GPU hardware. Significant atomic operations to serialise execution result in poor performance compared to the serial A1 option.

Iteration A13 – Naïve SP Flow8
Unstable results so not reported.

Iteration A14 – High-Low SP Flow8 on GPU
Inspects the size of a trip and accumulates into two SP 'buckets', one for low size numbers and the other for high size numbers. Model results are more stable with minimal loss of precision using multiple 32-bit summations equating to 0.000022% total error. A significant speedup compared to A1.

Iteration A15 – Improved DP Performance by Reducing Atomic Contention
The use of atomic operations is 'minimised' by processing a single link of each trip at a time, sorting the required data and using parallel reduction, reducing the number of atomic operations required. A significant speedup compared to A1 was achieved with accurate results.

4.4 SATALL integration development iterations

The optimised Bellman-Ford algorithm implementation, including updates for route choice and flow accumulation, plus optimised integration, allowed three further development iterations to be created and tested within this project.

The list below details each iterative change of the SATALL code, now referred to as the new SATGPU program, and **Figure 10** and **Figure 11** shows the serial and multi-core speedup comparison for Xeon Server (12 cores / 24 threads) Titan X (Maxwell, 3072 cores, 1075 MHz, 12GB GDDR5).

Figure 12 and **Figure 13** shows the serial and multi-core speedup comparison for PC i7 (4 cores / 8 threads) Titan X (Maxwell).

Iteration A16 – A11 with OpenMP Flow Computation on CPU
Hybrid version of GPU for route choice and flow accumulation, plus a CPU OpenMP multi-core for outer user class loops (part of the supply segment loops). However, the results unavailable due to unresolved integration issues.

Iteration A17 – A15 with Improved DP Flow Compute
First SATGPU release candidate and provides best DP results.

Iteration A18 – A14 with Improved accuracy SP Results
Second SATGPU release candidate and best runtime performance using SP with improved flow accumulation accuracy and stable model results.

Figure 10 H1 - Server Xeon Titan X SATGPU serial runtime comparisons (total SATASS + SATSIM assignment integrated code)

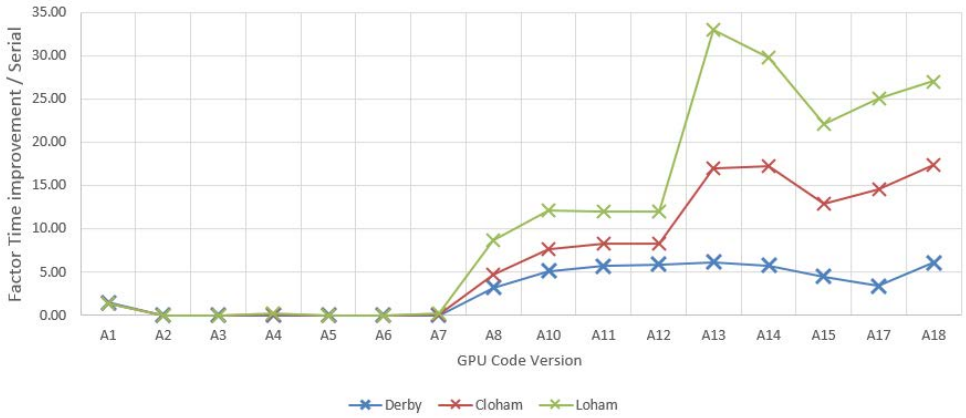


Figure 11 H1 - Server Xeon Titan X SATGPU multi-core runtime comparisons (total SATASS + SATSIM assignment integrated code)

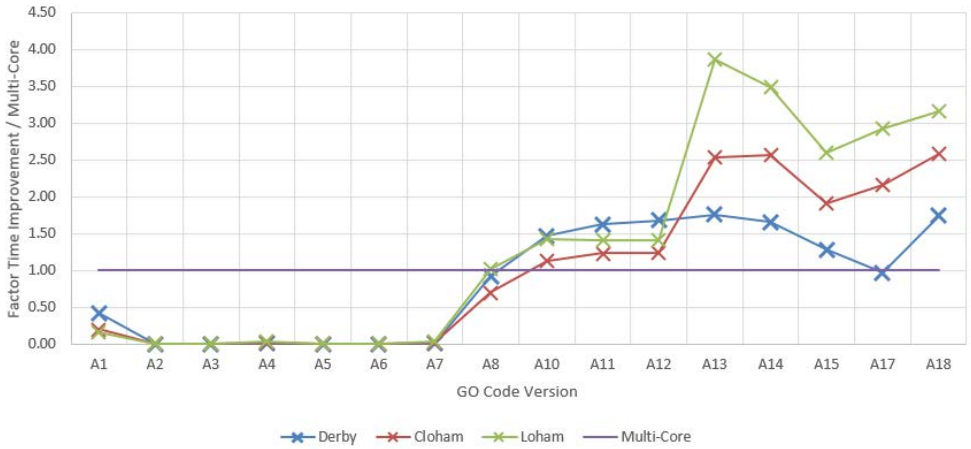


Figure 12 H2 - PC i7 Titan X SATGPU serial runtime comparisons (total SATASS + SATSIM assignment integrated code)

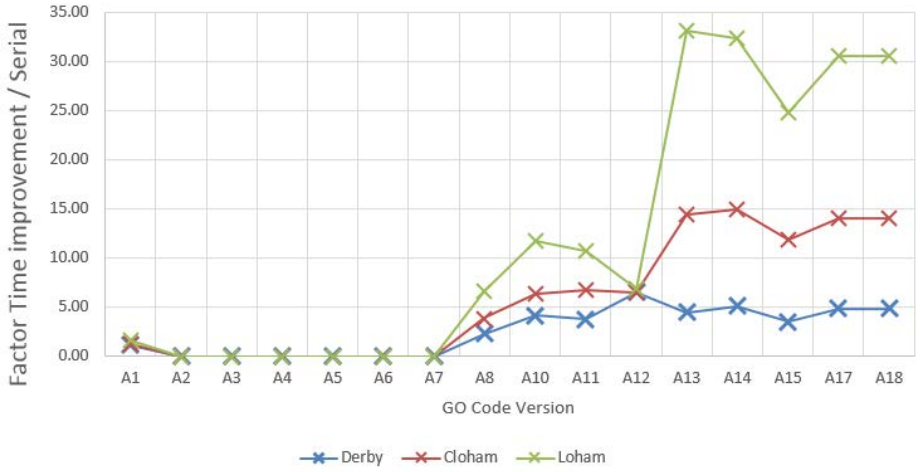
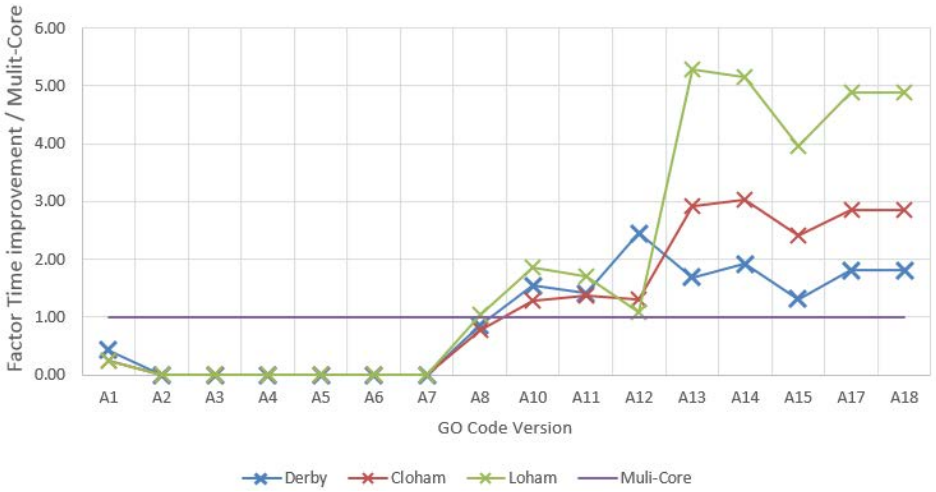


Figure 13 H2 - PC i7 Titan X SATGPU multi-core runtime comparisons (total SATASS + SATSIM assignment integrated code)



4.5 Software Testing

The testing involved comparing a number of results and checking that certain rules were passed successfully. Model stability was important, which was primarily measured using convergence stability, and helped show how the software could replicate previous results. This was a particular issue for the flow accumulation as the use of SP, or DP emulation, was known to make the model outputs unstable.

Furthermore the Bellman-Ford algorithm when implemented in parallel is not deterministic for networks with multiple, equal-cost shortest paths. When a network contains multiple routes from A to B with equal cost paths there is no guarantee that one route will be selected over the other (or should there be). This could be resolved for single link variance with some impact on performance by checking the link ID of equal cost links and selecting the lowest / highest value. This does not resolve routes with multiple link routes with equivalent costs - the alternate route with an equal cost path will not be considered due to how the frontier propagates one link at a time. A way to resolve this issue was not found without a significant impact on performance. However, if the model was fully converged then stable results compared to the reference case were achieved. No convergence or stability issues were encountered except for the use of SP in the flow accumulator. **Table 2** below contains the CLoHAM convergence summary for software version A15.

Table 2 Software version A15 CLoHAM convergence summary

Measure / Criterion		Multi-Core (Reference)	A15 (GPU)	%Diff
Convergence	Stability (%Flows)	98.5%	98.8%	
	Stability (%Delays)	99.3%	99.4%	
	Proximity (%GAP)	0.009%	0.009%	
Summary Statistics	Total Distance (pcu-km)	2,522,793	2,522,916	0.00%
	Total Time (pcu-hrs)	114,903	114,918	+0.01%
	Total Delays (pcu-hrs)	4,370	4,369	-0.02%

It was important to not only benchmark each software version against a reference case but to also repeat a full model validation. As such an existing LoHAM calibration and validation analysis was successfully repeated as shown in **Table 3** below.

Table 3 Repeat of LoHAM validation

Measure / Criterion	Aspiration	Multi-Core	A15	Diff

		(Reference)	(GPU)	
Links - GEH <5	85%	64%	64%	0%
Links - GEH <7.5	85%	78%	78%	0%
Links - DMRB Flow Criteria	85%	74%	74%	0%
Screenline - Flow Difference <5%	85%	90%	90%	0%
Enclosure - Flow Difference <5%	85%	94%	96%	+2%
Mini screenline - GEH <5	85%	91%	91%	0%
JT Routes - Time Difference < 15%	85%	92.1%	92.6%	+0.5%
Links - GEH <5	85%	64%	64%	0%

5. NEXT STEPS

5.1 Potential for further runtime improvements

During the commissioning and early stages of this project it was confirmed that to achieve a step-change runtime reduction that the SATALL code needed to become massively data parallel, and that the most likely future platform to provide continued improvement in computational speed was through the use of GPU-based technology. This first phase therefore investigated whether there is potential to use GPU-based technology to improve the runtimes of a traffic assignment model. This has been achieved by moving sections of the code that contain the path build onto a single GPU device. The areas for further investigation are discussed below and include:

- further algorithmic optimisation;
- scaling and improvements in hardware; and
- investigating other serial processing within the existing code.

5.2 Future algorithmic optimisation

Potential future optimisations have been identified but not implemented during this first phase, as listed below.

- **SSSP priority queue** - process network nodes in two stages, with the likely-update nodes before less likely (see Gunrock Delta stepping).

- **SSSP for origins with trips** - an optimisation from the original Load-It routine. If an origin has no trips, do not compute the SSSP. However, this scenario was not present in any of the test models.

- **Additional asynchronous compute / data transfer** - perform more operations asynchronous to other work.

- **Reduced 'pinning' of memory** - pinning memory has a cost. If the flow data could be pinned once and only once this would improve performance. Initial investigations resulted in model instability.

- **Block-level sorting for flow accumulation** - flow accumulation currently suffers from poor memory access patterns. A global sort to improve this costs more than it gains, but doing a block level sort should improve performance, and for both DP and SP variants. This only applies to Maxwell based GPUs.

- **Improved SP** – the performance offered by SP flow accumulation compared to DP on Maxwell based GPUs and earlier is considerable. Techniques to reduce the loss of precision could be implement to produce a SP solution with no or minimal loss of precision compared to using DP (Kahan Summation, etc.).

- **Improved support for devices with limited memory** - large networks may suffer runtime constraints on GPUs with fewer cores / memory than tested during this initial project. The flow accumulation method may need modifying to account for very large trip matrices on these smaller devices, by batching the number of trips being traced concurrently.

- **Minimise data transfer** - data is copied from host memory to GPU memory many times throughout the application runtime. This data transfer over the PCI express port is relatively slow and costly in terms of runtime. On GPUs with sufficient memory the majority of these data transfers could be performed once and only once, greatly reducing the time spent transferring data.

- **Multiple GPUs for stochastic simulation** – parallel SSSP for alternate route networks is non-deterministic. By launching the same simulation many times on multiple devices the costs of file input /

output, etc., are minimised, while allowing multiple runs to be aggregated / analysed.

- **Process multiple user classes concurrently** – in the test networks, up to thirteen user classes are processed serially one after the other. Each user class is essentially independent when the flow is accumulated, so multiple user classes were processed concurrently significant gains in performance could be achieved (potentially up to 'n' user classes provided sufficient memory and compute is available).

- **Modified spider network generation** - the generation of spider networks from non-spider networks could be modified to produce networks which are better suited to GPU based processing. Denser networks with a lower 'diameter' will result in increased performance.

5.3 Scaling and improvements in hardware

This first phase has demonstrated the use of a single GPU to accelerate the assignment phase of SATALL, optimised for the Maxwell Titan X. These new data-parallel methods can be scaled up to process more data concurrently and, where required, scale across multiple GPUs to take advantage of the increasingly large computational power available at relatively low cost.

For the smaller models a single user class does not fully occupy a GPU. Launching the assignment operations for multiple user classes on a single GPU concurrently would further reduce the run-time of the application and also pave the way for subsequently undertaking multiple user classes on multiple GPUs. Using multiple GPUs for a single application would provide benefit to larger models which occupy a much greater portion of the resources of a single GPU. This could reduce the runtime of assigning all the user classes to the time taken to assign the longest running user class. On a single GPU each user class would be assigned to its own CUDA streams, which the CUDA runtime intelligently handles on a single GPU. Across multiple GPUs built-in CUDA functions would select the appropriate device for each user class prior to the assignment algorithm launch to load-balance the system.

For very large models, greater in size than the largest known SATURN network used in this project, the assignment process for a single user class may not fit on a single device, although the processing would be batched. Changes to the assignment code could exchange data between multiple

GPUs at the beginning and end of assignment iterations as they separately perform the assignment on different subsets of the same user class.

A single PC workstation will typically support two to four GPUs, although configurations of up to eight are available. This approach to scaling therefore becomes limited by the number of user classes and size of the model, limiting the runtime reductions when multiple GPUs can be used asynchronously for parallel user classes on multiple GPUs. This issue may not be reached, as for LoHAM and RTMs the largest single user class comfortably fits on a single Titan X GPU device. However, if models bigger than LoHAM are encountered then further scaling could be achieved by accessing a large compute cluster.

At the cluster-level, the programming model must change slightly as communication between the different nodes of a cluster must be performed explicitly. Message Passing Interface (MPI) would be used to communicate data between different GPUs, but otherwise the methodology outlined previously would hold true. As well as the benefit of improved performance by splitting user classes across many GPUs, an MPI-enabled multi-GPU implementation would allow SATGPU to handle network sizes far larger than those currently available, and are planned for the foreseeable future.

A performance limiting factor for the flow accumulation code in DP is the atomic add operation in DP. Up to the Maxwell Titan X architecture the DP atomic add is not implemented at the hardware level. Instead it must be implemented in software, the method of doing so is not ideal and the issue compounds as more atomic operations are required, i.e. larger models. This has a significant negative impact on performance and therefore a much more complex flow accumulation method was implemented for this phase. The newer Pascal GPU architecture combined with the latest version of CUDA (CUDA 8.0) does include this instruction at the hardware level. **This offers significant performance gains compared to the solution used for Maxwell based GPUs, with early work suggesting a further assignment speedup of up to 2x compared the Maxwell GPUs already demonstrated.**

5.4 Investigating other serial processing

Amdahl's law correctly identifies that the serial processes within the SATGPU program run will constrain the overall runtime improvements. This phase of the project has focused on the traffic assignment as the highest computational element of the existing SATALL runtime. However, current runs of the RTMs are now indicating that the junction simulation are equalling or even exceeding

the runtimes of the assignment. So a clear focus for future development is to also parallelise the simulation code. This could start with CPU parallelisation but, following this project, it seems more likely that a GPU implementation could provide more significant longer term improvements.

Achieving massive data parallelisation in the simulation code could require algorithmic changes. However, starting with a simple implementation on GPU could provide quick and significant runtime improvements, and would also provide the building blocks for further improvements.

6. CONCLUSION

There is no doubt that, in an attempt to find efficiencies, authorities are moving to larger transport models that can be used across many schemes, and that further modelling requirements for bigger and more complex models will continue. Previously this has not been a practical option due to data and model runtimes. With the advent of generic maintained datasets, and contemporary datasets like mobile phone data and GPS data, this information is now enabling the creation of larger models, both regional and national. However, whilst there has been significant improvements in computer hardware power and affordability, for example ‘blade’ servers, as well as significant improvements in CPU based multi-core parallelisation, model runtimes still remain a major constraint for the growing size of models.

Our research has indicated that, irrespective of the hardware, the first task of making the software massively data parallelisable is achievable. This is a significant task as the algorithms that may have been the most efficient for serial, and task parallelism, may not be the best for data parallelisable. This project has shown this is possible by successfully replacing the D’Esopo-Pape path build algorithm with the Bellman-Ford algorithm. The project has demonstrated that through a number of optimisations the Bellman-Ford algorithm can achieve the require step-change in assignment modelling runtime. And this is irrespective of the wasted computation, which is now affordable due to the abundance of threads available on the GPU device.

The optimisation of implementing the GPU-based technology was bespoke for each instance of the path build algorithm. This included exploring the best options for the computationally slow flow accumulation using either: DP numbers and achieving a high level of stability; or emulating the flow accumulation with SP or approximation of DP, and achieving varying levels of stability. The outcome of the implementation of Bellman-Ford is not only

highly stable but has also been shown to successfully reproduce the existing LoHAM validation, one of UK's largest models covering Greater London.

With the principle established that data parallelism is now possible, the problem then becomes about scaling across multiple GPUs, and importantly that the copying of data across the PCI express port 'bottle neck' can be achieved asynchronously across multiple GPUs. The larger the model the more efficient this transfer becomes, and the denser the network the more efficient the Bellman-Ford implementation becomes, meaning this approach is well aligned to meeting the runtime challenges faced by the direction of change of increasing model size and complexity.

This report indicates that the shippable SATGPU program from this project, with a single Maxwell Titan X for the largest models, can be expected to achieve up to 5x faster than PC based multi-core and up to 30x faster than serial. And the advances in the new Pascal Titan X (August 2016) further demonstrates that GPU-based technology is most likely technology to maintain Moore's law, and provide affordable and substantial increases in speed due to its energy efficiency and cost.

BIBLIOGRAPHY

Davidson, Andrew, et al. "Work-efficient parallel GPU methods for single-source shortest paths." Parallel and Distributed Processing Symposium, 2014 IEEE 28th International. IEEE, 2014

Głąbowski, M., et al. "Efficiency evaluation of shortest path algorithms." AICT 2013, The Ninth Advanced International Conference on Telecommunications. 2013

U. Pape, Implementation and efficiency of moore-algorithms for the shortest route problem, Mathematical Programming 7 (1) (1974) 212–222

Van Vliet, D. Hall, M.D. and Willumsen, L.G. (1980) SATURN - A Simulation Assignment Model for the Evaluation of Traffic Management Schemes, Traffic Engineering & Control, 21, 168-176, April 1980.

Van Vliet, D. (1978) Improved Shortest Path Algorithms for Transport Networks. Transportation Research, Volume 12, pp. 7-20. Pergamon Press 1978.

Wang, Yangzihao, et al. "Gunrock: A high-performance graph processing library on the GPU." Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. ACM, 2016

Wright, I. Xiang, Y. Van Vliet, D. Bar-Gera, H. Boyce, D. (2010) The Practical Benefits of the SATURN Origin-based Assignment Algorithm and Network Aggregation Techniques. Proceedings of the European Transport Conference 2010