Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 151 (2019) 409–416

Procedia Computer Science

www.elsevier.com/locate/procedia

The 10th International Conference on Ambient Systems, Networks and Technologies (ANT)
April 29 – May 2, 2019, Leuven, Belgium

High Performance Computing applied to Dynamic Traffic Assignment

Willem Himpe[a,c]*, Romain Ginestou[b], M.J. Chris Tampère[c]

[a]*Transport and Mobility Leuven, Diestesteenweg 57, 3010 Leuven, Belgium*
[b]*ENSTA, PariTech, Boulevard des Maréchaux 828, 91762 Palaiseau Cedex, France*
[c]*KU Leuven, L-Mob Leuven Mobility Research Center, Celestijnenlaan 300A, 3001 Leuven, Belgium*

Abstract

This paper aims at applying High Performance Computing to Dynamic Traffic Assignment. The latter are a class of algorithms designed for accurate traffic simulations on networks. The motivation to bring together these two distinct fields of study is primarily to enable traffic engineers to tackle real-size traffic problems on large networks, using the powerful approach of Dynamic Traffic Assignments. Various insights and guidelines are presented, as well as their impacts on computation time and memory footprint. The main contribution of this paper is the method of domain separation, which is applied to cut a road network into regions. It enables parallel computing in a cluster environment, whilst ensuring a consistent synchronization across the global network. Additional speed ups are achieved by balancing the amount of operations within a region and the quality of the overall result. It is hoped that this study can bring interest for researchers of both domains, to further improve the efficiency of traffic algorithms for realistic simulations.

© 2019 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)
Peer-review under responsibility of the Conference Program Chairs.

Keywords: Traffic Simulations; Link Transmission Model; Dynamic Traffic Assignment; High Performance Computing

* Corresponding author. Tel.: +32 16 95.71.52
E-mail address: willem.himpe@kuleuven.be

1877-0509 © 2019 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)
Peer-review under responsibility of the Conference Program Chairs.
10.1016/j.procs.2019.04.056

1. Introduction

High Performance Computing (HPC) refers to a set of practices and techniques aimed at delivering and optimizing vast amounts of computing power to solve large problems in science, engineering or trade. In other words, HPC aims at harvesting the full computing potential of a computer (or a set of computers) to solve a resource-demanding problem. This research primarily aims at bringing some insights from the HPC realm into Dynamic Traffic Assignment (DTA) research. An overview of recent advances is found in [1]–[3]. Different kinds of optimizations have been performed throughout this study on the entire DTA procedure. These improvements fall in three categories:

- Core optimizations which include smart data structure, cache friendly code (low level optimizations)
- Parallel computing which aims at spreading the computation across multiple computation units to be able to cope with different parts of the simulations at the same time
- Operation savings taking into account the inner workings of each algorithm to find ways to reduce the amount of operations needed, while keeping the simulation's output satisfying

1.1. Low level optimizations

The use of a low level programming language (like C or even C++ as it is the case in this study) helps design efficient software by neatly managing memory and crafting code close to what the resulting program's binary code looks like. Many optimizations mainly relying on developing cache-friendly code to minimize memory access time and enable vectorization, as well as best-suited data structure choices. The underlying algorithms for DTA (network loading and shortest path) often depend linearly on each of the problems dimension (number of time intervals, number of nodes or links, number of destinations in the network).In this sense, the main challenges are memory management and best-fitted data structures, since the computations process large amounts of memory to output the simulation results.

1.2. Domain decomposition

HPC often advocates parallelization to spread the computing burden across multiple cores of multiple computers, all running at the same time, which leads to potential speedups. When it comes to making simulations on objects having spatial dimensions such as solid bodies when studying mechanical deformation, liquids in fluid dynamics, or even a road network in traffic flow studies, it is often possible to apply the simulation on parts of these elements separately, and stitching the results together afterwards. This technique is called domain separation. One can consider the interest of this technique to perform parallel computing, as it makes it possible to attribute one region to one computer core and run all the simulations at the same time. Domain decomposition has been of great interest for this study.

1.3. Fixed-point iterations

The inner workings of the developed DTA algorithm allows for arbitrary precision levels by repeating computations and/or limit processing time by truncating iterations. Additionally, methods that calculate efficiently repeated evaluations are important for many applications as often multiple scenarios have to be analysed. An intuitive explanation for the importance of such type of algorithms in traffic networks is to consider the outcome of different scenarios. A part of the calculation scheme for each scenario might be redundant because the traffic states are similar or even exactly the same for some regions in space and time [4]. This is accomplished by developing iterative formulations that allow for a warm start such that the computations can be initialized with previous results.

2. Dynamic traffic models

Mobility plays a key role in modern societies. Road networks provide the infrastructures for transportation of goods and people, but the growing demand often reaches the limits of these networks. An entire research field is

dedicated to studying, simulating and optimizing these networks, in order to face current and future challenges. Dynamic traffic simulations are being actively developed to offer a fast and reliable means to simulate transportation networks. This research study focuses on the computational aspects of such dynamic simulations, which are usually labeled Dynamic Traffic Assignments (DTA). The core components of the DTA presented throughout this paper are derived from the MatlabTrafficToolbox[†].

2.1. Dynamic User Equilibrium

The DTA aims at simulating a real-life situation in a given road network. Within a network, the individual choices of all the vehicles driving on its roads affect travel times and may for instance cause traffic jams. Drivers respond to the foreseeable congested state of the network by evaluating their routing decisions. Let's take the point of view of a single vehicle inside a road network. A driver is assumed to choose (in a selfish manner) a path which will minimize the time spend in the network, that is to say arrive at its destination as soon as possible. The shortest path chosen by a given driver (shortest in terms of time spent in the network) might also turn out to be the shortest path for a large number of other vehicles, hence causing congestion on the road, and delay. The driver may therefore have to reconsider its route choice, and take another path, which can change congestions patterns as well. The iterative nature of the relationship between route choice and network load has to be taken into account to accurately simulate dynamic traffic into a network. The optimal result is achieved when equilibrium [5] is reached between the route choice and the network loading, that is, when choosing shortest paths in a network do not change the network load in terms of traffic conditions, and when computing the network load does not affect driver behaviors, which remain on their current shortest path. DTA simulates how traffic flows in a road network by reaching a Dynamic User Equilibrium (DUE) between route choice and network loading. These algorithms are generally organized around these two fundamental pillars:

- A dynamic network loading (DNL) algorithm: calculates the traffic conditions given the inflow of drivers at the entrance of a network from which we can infer the travel times along any road in the network
- A route choice algorithm, which translates the arrival time within the network into optimal paths for the drivers to reach their destinations

A final important component of a DUE algorithm is what kind of solution algorithm is used to find the equilibrium. Typically fixed-point algorithms (such as the gradient decent method) are used for this purpose [6]. The scope of this paper is limited to HPC optimizations for DNL computations. However, all of the presented techniques can be used to develop a full DUE methodology as will be shown in a future publication.

2.2. Dynamic Network Loading

The Dynamic Network Loading algorithm takes a crucial place in the Dynamic Traffic Assignments. Given the demand in a network (that is to say the number of vehicles wanting to enter the network at any given time), and where the vehicles flow (specified by turning rates towards specific destinations at each intersection), this algorithm simulates the evolution of the network's congestion state through time. This research study uses a specific class of Dynamic Network Loading methods known as Link Transmission Models (LTM). These algorithms offer a macroscopic approach based on Newell's simplified theory of kinetic waves [7]. More specifically, the LTM used throughout this study has been developed by[4]. This version of LTM handles simulations with large time intervals while still computing accurate congestion patterns. It is an iterative scheme that offers the ability to be warm started, which is of huge interest for high performance computations.

[†] MatlabTrafficToolbox is a Matlab package developed by the KULeuven. It is made available as open source package here: https://www.mech.kuleuven.be/en/cib/traffic/downloads

2.2.1. Network topology

In order to understand how the LTM works, one first need to consider the network topology used for the simulations. The figure below shows a very simple example of a network that could be used in a DTA. It is structured as an oriented graph, with a set of nodes and a set of links between these nodes. Some special nodes are labeled either as Origin (O_1 and O_2 here) or as Destinations (D_1 and D_2 here). These nodes are the network's entry or exit points. The vehicle flows from the origin to the destination nodes.

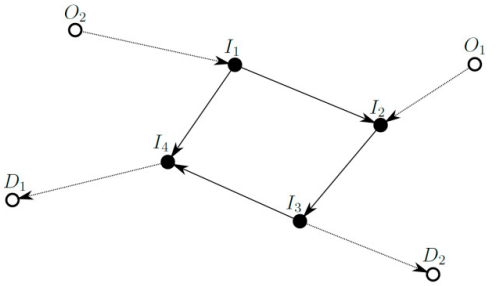


Fig. 1. network sample with 2 origins and 2 destinations, 4 directed links between the internal nodes and 4 connectors connecting the origins and destinations to the internal nodes.

The LTM computes cumulative vehicle numbers (CVNs) for every link's upstream and downstream end at each time step of the simulation. The CVNs represent the total number of vehicles that have passed through the respective ends of a given link since the beginning of the simulation, distinguished by destination. Upstream and downstream CVNs are not independent from each other, but rather impose constraints on each other representing the physics of traffic and congestion propagation.

In downstream direction, the upstream CVN imposes a constraint on the downstream CVN of the same link, in that no more vehicles can have passed the downstream link end, than there have been vehicles entering some time earlier. This time lag is in this case the minimum time to cross the link. Likewise, if the outflow (represented by downstream CVN difference in a time interval) of a link is constrained (e.g. congestion in the next link spills back, or the node itself reduces outflow e.g. through means of traffic lights), this may impose constraints on the amount of traffic capable of entering the link (i.e. upstream CVN) sometime later. Newell's simplified kinetic wave theory quantifies this constraint on upstream CVN as the downstream CVN plus an increment equal to the maximum number of vehicles that fits inside the link. The time lag is defined as the time to cross the link with the fastest congested wave speed.

A complete discussion of the fundamentals of first-order traffic flow theory and different solutions schemes can be found in most popular traffic engineering handbooks [8], [9].

2.3. LTM's iterative scheme for solution finding

The LTM works by computing the CVNs for each time interval moving forward in time. The inflow of vehicles entering the network is added to the CVNs of the origin nodes. To update every node inside the network, the algorithm needs to compute for every incoming and outgoing link the maximum CVN that is allowed by the internal link dynamics (constraint propagation according to Newell's simplified theory). These are called sending and receiving flows for incoming and outgoing links respectively. Next the node model decides on how sending flows are transferred over the node while complying with receiving flow restrictions and local route choice behavior. The route choice is expressed implicitly as destination based splitting rates requiring CVNs to be composed of multiple commodities per destination. The final dimension of the CVN data structure is defined by the number of simulation intervals, the number links and the number of destinations. The LTM has a nested loop structure. One loop iterates through time and encompasses a nested loop iterating until convergence of CVNs, within which a loop iterates over all of the network's internal nodes. Without getting into the details, here is the overall structure given as pseudo-code below. Note that the Node Sync Call is related to the parallelized code that is described in section 3.4.

```
Do for all time intervals
        Deal with origin nodes
        Do until convergence of CVNs
                -> Node Sync Call <-
                Do for all active internal network nodes
                        Compute sending and receiving flow
                        Apply node model
                        Update CVNs
                Update active nodes
        Deal with destination nodes
```

2.4. Warm starting

In order to avoid unnecessary computations, the LTM only considers a set of active nodes to be updated within a region. The activation process works as follows: all origin nodes are activated in the first iteration; a considerable change to a CVN at one link end leads to the activation of the node at the opposite link end; activated nodes are to be recalculated in the next iteration. A node is only activated if the total absolute amount of CVNs changes at its respective link ends is large enough.

In such a way only the significant changes in CVNs are propagated throughout the network. From a programming point of view, a threshold is introduced in the LTM to determine whether or not a node should be activated. A big threshold will lead to fewer nodes being updated, since the propagation of changes will be stopped soon after it fades out and goes below the threshold. On the other hand, a very small threshold ensures an almost exact computation but will lead to the activation of a lot of nodes affected by minor changes.

The activation process makes it possible to warm start the LTM procedure. That is, rather than starting from zero CVNs, one can input CVNs resulting from a previous simulation. Since the CVNs are closer to an optimal solution, the activated nodes rapidly come under the activation threshold, and the CVN changes don't propagate far, thus making iterations converge faster and reducing the overall number of computations.

3. A balanced method to distribute the network over regions

Next the method designed to cut the network and perform the DTA procedure on a multi-core environment is presented. Beyond the mere algorithmic optimizations that have been presented so far, one of the main focuses of this study has been to develop techniques in order to run the entire DTA on several computers within a cluster. We call these computers cluster nodes. Here are some of the motivations behind the need to support cluster computing:

- A single computer has a limited number of threads on which the simulation can run in parallel. Cluster computation can spread the computation across more threads on several machines, hence giving a potential speed boost, under some conditions.
- A single machine has a limited memory capacity. Some huge networks may require several dozens of gigabytes of RAM to store disaggregated CVN values and turning rates per destination. When the amount of memory on one machine is not sufficient to run the simulation, turning towards cluster computing is mandatory.

Domain separation aims at cutting the network into multiple regions, one for each cluster node, in order for each of the cluster nodes to simulate only part of the entire network, hence bearing only a fraction of the global memory and processing cost. The network cut needs to satisfy some constraints in order for the simulation code to be easily maintainable, and for the simulation to be efficient.

3.1. Scale-agnostic computations

The idea behind domain separation is for every cluster node to be autonomous when performing the simulation within its designated region. One of the main focuses of this study was to make sure the cluster computing

specifications would not disturb the DTA algorithm's inner workings. The scale of the network in a region, be it a single region or the entire global network, should not play a role in how the algorithm behaves. The cluster computing code for node synchronization is present at one special point in the algorithm to fetch data from the other nodes rather than from local memory. In this sense, cluster computing is only used as a means to share memory, but has no consequences in terms of algorithm design.

3.2. Efficient cutting strategies

There is a transfer of information across the region's boundaries to keep the global result consistent. In other words traffic flow theory also has to hold on the roads between two regions. This information exchange has a huge cost compared to local memory access, because it has to be passed from one computer to another. Two key aspects are to be taken into account to design efficient cutting strategies:

- The amount of information exchanged at the boundaries should be as small as possible to reduce the time it takes for the data to go from one cluster node to another. The number of boundary links – that is, the number of links going from one region to another different region – should be minimized.
- The regions should be equivalent in terms of computation time requirements. A global network iteration is over when the every region has finished its computations. Afterwards the information is shared across the boundaries and the next iteration can start. One can understand that if all regions are balanced according to computational burden, idle time per region is minimal and therefore overall computation time is optimal.

These two criterions are related to the sparsest cut graph theory problem. This problem is NP-Hard, and the algorithms developed so far [10] are beyond the scope of this study. A simpler approach will be undertaken here. We propose the following criterion to be minimized in order to come closer to a balanced network cut:

$$\gamma = \frac{\sum_r L^r_{boundary}}{\prod_r N^r} \tag{1}$$

Where $L^r_{boundary}$ is the number of boundary links of region r, and N^r the number of nodes in region r. This criterion's value decreases as the number of boundary links decreases. The denominator reaches its maximum value when all regions have an equal number of nodes. A heuristic is used to derive a balanced network cut by simply distributing the number of nodes equally over regions with a breath first method. The algorithm responsible for cutting the network makes several attempts (typically 10) to cut the network starting at random, and the configuration that minimizes γ is withheld. The method works on the global network by building R region sets as follow:

- R network nodes are chosen at random (the distribution can be uniform over all nodes, or depend on density, for instance), each of which is added to an empty region set.
- For every region, look for the neighbors of the node picked in the region set – that is, the nodes which share a link with the region's node. Pick the closest neighboring node and add it to the region's set. This neighboring node should not belong to any other region.
- Repeat the closest neighbor selection until all the network's nodes are part of one of the R regions.

3.3. Network merger for node synchronization

The network's regions are meant to be somewhat independent from each other. Even though we don't want a region to be aware of the entire network to be computed, the simulation itself needs however to be aware of the boundaries between the regions to be able to synchronize and share simulation results across the regions. A new type of node has been introduced in the network to identify the boundaries. The different types of nodes are therefore origin, destination, internal and boundary nodes. Boundary nodes are located on the region's periphery, and are redundant with the internal node just behind the boundary of the neighboring region, as illustrated in Fig. 2.

Two regions are depicted here. Node N1, N2 and N3 are all internal nodes of the blue region on the right. The blue region's boundary nodes are node N4 and N5, which are in fact internal nodes of the red region on the left. At the same time, the boundary nodes of the red region are N2 and N3, internal nodes of the blue region.

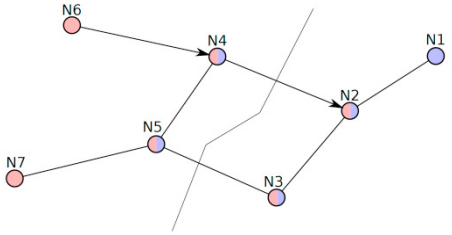


Fig. 2. How nodes are shared across a two-region boundary

These nodes are meant to be considered as origin and destination, in the way that their value is imposed by the simulation results of the neighboring regions. Therefore, when the red region is done computing its CVNs, it can send the CVNs of node N4 and N5 to the blue region, which has N4 and N5 as boundary nodes. The CVNs coming from the red regions are considered as imposed inputs, and the blue region's simulation can take them into account for its current simulation iteration. This process is in all aspects similar to what's happening on the origin and destination nodes.

A network merger is needed to express global indices of the network elements into local indices of a region. The demand is loaded and cut so that only the origin nodes within a given region act as the region's sources. The destination based structure of the DNL requires every region to account for all destinations in the global network although only a limited set is available as sinks in each region. At the very end of the simulation, the computed regional CVNs are translated into global indexed CVNs and gathered into one single file that corresponds to the global network. These results can therefore be interpreted without having to be aware of how the network was cut.

3.4. Cluster-computing-ready architecture

The C++ MPI library [11] was used throughout this study to build a cluster computing-ready software, able to manage a multi-region simulation based on the network cut discussed before. This library allows to seamlessly integrate parallel computing to a piece of software by running this same software as multiple processes on several cores and several computers, and exchanging information between these processes using dedicated synchronization calls. Every single process launched in a cluster using the MPI library is aware of the total number of processes currently running, as well as its own process id. Generally speaking, the DTA's algorithms are applied to each separate region, and then the information at the boundaries is shared among the regions.

A communication routine is used in order to synchronize the regions between the MPI processes. This routine will fetch information provided by all the MPI processes, and send them to every other process. Every region has access to all the boundary information, and can therefore pick the relevant results to go on with the simulation.

The synchronization points need to be carefully placed within the algorithms' structure in order for the simulation to stay consistent even though part of the regions' data is fetched from other processes. Yet, having the lowest number of MPI communication calls is best for performances. We therefore wish to place the calls inside the main loop, and not in the nested loops that are often repeated. However, we also want to keep the simulation consistent. We don't want to change the algorithm's logic and inner workings. The regions therefore need to be synchronized nearly as often as in a global simulation where the entire network would have been computed all at once. The synchronization call happens in the LTM inner iterations, right before iterating on the internal nodes (see pseudo code before).

This structure makes sense, because the synchronization call is in fact a network node update, applied to boundary nodes using data coming from neighboring regions. The cluster-computing-ready code is not invasive given this algorithm structure. It only interacts with a certain type of nodes (at the boundary) which is not considered by the rest of the LTM (going through internal nodes only), hence avoiding any conflicts between the different parts of the procedure.

4. Preliminary results and outlook

The figure below summarizes the observed speed-ups over different setups of clusters and regions in a typical network (23457 links; 11097 nodes; 1061 origins; 1061 destinations). If there are more regions present than nodes on the clusters, multiple regions are evaluated on a single node (using multiple threads). The results are presented relative to a single threaded program running on the global network. The outcome is as expected with clear reductions in computation time for code running in multiple threads on a single node represented by the diagonal elements. All elements below the diagonal relate to results involving multiple nodes on a cluster. Also here computational gains are made although also the limitations due to communication overhead become visible.

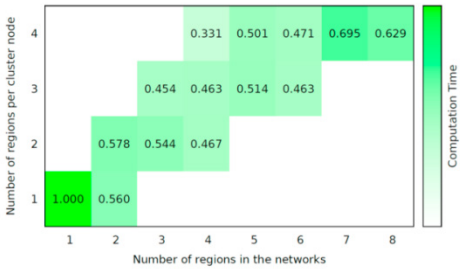


Fig. 3. Relative computation times heat map for different cluster configurations.

In the previous result the synchronization call happens in every LTM inner iteration. In future research we will look at balancing the amount of synchronization calls with internal computations of a region. This could be done by syncing boundaries only after a number of iterations, allowing for emphasis on the accuracy of a result within a region over the consistency across boundary links. In the extreme case regions could be evaluated and managed completely separately only to be synchronized when scenario's ask for their interaction.

This also relates to the network cutting strategy. So far, this method cuts regions by distribution all of the network's nodes evenly over the regions. Running a short simulation on beforehand can give some information about how active some areas of the network are, and one might consider grouping those nodes into a single region. The analysis of such regions on itself is a useful network tool to identify (hierarchical) structures that show strong coupling. But it could also lead to balancing of computations in more dimensions with some regions having less nodes but performing more iterations before syncing or vice versa.

In conclusion, it is hoped that this study can bring interest for researchers of both the high performance computing and traffic simulation domains, to further improve the efficiency of traffic algorithms for realistic simulations, and to illustrate a rich case study for highly optimized computations.

References

[1] A. Aboudina, I. Kamel, M. Elshenawy, H. Abdelgawad, and B. Abdulhai, "Harnessing the Power of HPC in Simulation and Optimization of Large Transportation Networks" *IEEE Intell. Transp. Syst. Mag.*, vol. 10, no. 1, pp. 95–106, Spring 2018.
[2] Y. Qu and X. Zhou, "Large-scale dynamic transportation network simulation" *Transp. Res. Part C Emerg. Technol.*, vol. 75, pp. 1–16, Feb. 2017.
[3] W. Turek, L. Siwik, and A. Byrski, "Leveraging rapid simulation and analysis of large urban road systems on HPC," *Transp. Res. Part C Emerg. Technol.*, vol. 87, pp. 46–57, Feb. 2018.
[4] W. Himpe, R. Corthout, and M. J. C. Tampère, "An efficient iterative link transmission model," *Transp. Res. Part B Methodol.*, vol. 92, Part B, pp. 170–190, Oct. 2016.
[5] J. G. Wardrop, "Road Paper. Some Theoretical Aspects of Road Traffic Research" 1952, vol. 1, pp. 325–362.
[6] G. Gentile, "Solving a Dynamic User Equilibrium model based on splitting rates with Gradient Projection algorithms," *Transp. Res. Part B Methodol.*, vol. 92, pp. 120–147, Oct. 2016.
[7] G. F. Newell, "A simplified theory of kinematic waves in highway traffic, Part I: General theory," *Transp. Res. Part B Methodol.*, vol. 27, no. 4, pp. 281–287, 1993.
[8] E. Cascetta, *Transportation Systems Analysis*, vol. 29. Boston, MA: Springer US, 2009.
[9] M. Treiber and A. Kesting, *Traffic Flow Dynamics: Data, Models and Simulation*. Springer, 2012.
[10] S. Arora, S. Rao, and U. Vazirani, "Expander Flows, Geometric Embeddings and Graph Partitioning," *J. ACM*, p. 30, 2009.
[11] E. Gabriel *et al.*, "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, vol. 3241, Springer Berlin Heidelberg, 2004, pp. 97–104.