Available online at www.sciencedirect.com

ScienceDirect

Transportation Research Procedia 00 (2018) 000–000

Transportation Research Procedia

www.elsevier.com/locate/procedia

International Symposium of Transport Simulation (ISTS'18) and the International Workshop on Traffic Data Collection and its Standardization (IWTDCS'18)

On the implementation of high performance computing extension for day-to-day traffic assignment

Wasuwat Petprakob[a], Lalith Wijerathne[b,*], Takamasa Iryo[c], Junji Urata[c], Kazuki Fukuda[c], Muneo Hori[b]

[a]*Department of Civil Engineering, The University of Tokyo, 7-3-1, Hongo, Bunkyo, Tokyo, 113-8654, Japan*
[b]*Earthquake Research Institute, The University of Tokyo, 1-1-1, Yayoi, Bunkyo, Tokyo, 113-0032, Japan*
[c]*Department of Civil Engineering, Kobe University, 1-1, Rokkodai-cho, Nada, Kobe, 657-8501, Japan*

Abstract

The ability to find near-optimal traffic assignments for a very large network within a few days time will substantially contribute to reduce economic losses following major earthquake disasters. However, lack of efficient numerical tools and methodologies to solve this NP-hard problem within a reasonably short time is a major challenge. This paper presents details of an HPC (High Performance Computing) enhanced system which is developed to address this need of finding near-optimal traffic assignment for large networks in a short time. The developed system is based on day-to-day algorithm and enhanced with a distributed memory parallel extension to accelerate the computation by utilizing computer clusters or supercomputers. Details of basic implementations, strategies to accelerate the serial computations and parallel scalability, and numerical examples to demonstrate the effectiveness of the proposed strategies are presented.

Keywords: day-to-day traffic assignment; post-disaster route guidance; high performance computing; domain decomposition

1. Introduction

Road network is a vital lifeline infrastructure which is supporting industrial and economic activities. Therefore, optimal utilization of functioning portions of a road network is essential to reduce the economic losses followed by major disasters, like Tokai, Tonankai and Nankai earthquakes which are predicted to damage major industrial zones of Japan. Recovery of road networks is time-consuming (e.g. 21 months to recover after the 1995 Kobe earthquake (Chang and Nojima (2001))), hence the reduced total delay will be substantial if traffic is optimally assigned. The

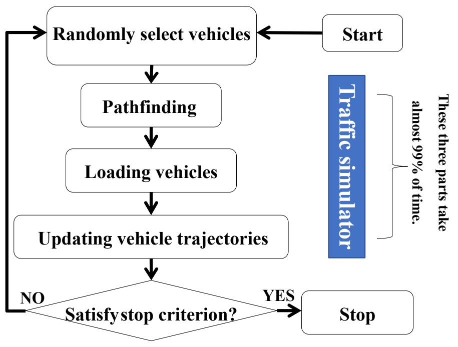
* Corresponding author. Tel.: Tel.: +81-03-5841-5699 ; fax: +81-03-5841-5699.
E-mail address: lalith@eri.u-tokyo.ac.jp

traffic assignment plans must be updated continuously, at least once in a few days, according to the repair progress of the damaged road segments. Lack of numerical tools to find near-optimal stable traffic allocation plans for large networks, like Kanto network of Japan, in a few days time is a major barrier. The current best implementation of dynamic traffic assignment can complete a single day iteration of New York network, with 25 million of vehicles, approximately within 1.5 hours using 128 CPUs (Thulasidasan and Eidenbenz (2009)); it does not scale well beyond 128 CPUs. At this computational capability, it may require more than a month to find a near optimal solution for the New York network, which is far longer compared to a few days period required for post-disaster traffic assignment. The aim of this research is to address this need by developing an HPC enhanced numerical tool to accelerate finding near-optimal solutions for large networks utilizing computer clusters or supercomputers.

Our HPC enhanced system is based on the day-to-day traffic assignment (D2DTA) with link transmission model (LTM), which is simple and has lightweight computational demand. Although D2DTA algorithm has low computational resource demand, it requires a large number of iterations, especially with large-scale problems, to converge. Therefore, it is essential to utilize HPC resources to find a solution for large networks within a short period of time. Several researches on the parallel implementation of dynamic traffic assignment are reported in literatures (Shen et al. (2008);Chabini et al. (2003);Attanasi et al. (2015);O'Cearbhaill and O'Mahony (2005)). However, none of them is sufficiently efficient and scalable to find an acceptable user equilibrium state of a large-scale road network in a reasonable short time. In this paper, we explore the strategies to reduce computation time and improve the scalability of D2DTA with the aim of applications in disaster recovery.

The rest of this paper is organized as follows. The second section provides a brief introduction to the D2DTA method. The third section presents details of HPC implementation and strategies for accelerating the computation and improving the scalability (i.e. ability effectively to use a larger number of CPUs to reduce compute time). The fourth section presents the numerical experiment to demonstrate the effectiveness of the proposed strategies to accelerate the computations.

2. Day-to-day traffic assignment



- *Randomly select:* Include a subset of any remaining vehicles from the input. Also, choose a random subset from the already included vehicles.
- *Path find:* Find the paths with minimum travel time for the above selected vehicles, based on the travel times estimated in the previous step.
- *Load vehicles:* The above selected vehicles are assigned to the links along the corresponding paths with the shortest travel time.
- *Update trajectory:* Simulate traffic flow using a suitable link delay model for handling congestions, and update the travel times.

Fig. 1: An overview of serial day-to-day traffic assignment.

A brief overview of day-to-day traffic assignment (D2DTA) is presented in this section for the sake of completeness. It is a daily experience that drivers change their paths according to their previous travel experiences in order to reduce their travel time (Smith and Wisten (1995)). The D2DTA algorithm mimics this drivers' behavior to find near-optimal traffic assignment. Numerical instabilities of iterative oscillation may be observed. Figure 1 summarizes the main steps involved in D2DTA.

Pathfinding and traffic flow simulation (i.e. updating vehicle trajectories) are the most computationally demanding components of D2DTA. In order to reduce the computational load, we use LTM for simulating traffic. Even though LTM is computationally light, still it consumes a significant time with large networks. Efficient parallelization of these two components is crucial to finding near-optimal traffic assignment for large networks.

3. HPC enhancement

A short introduction to the main steps of the developed distributed memory parallel day-to-day algorithm with LTM is presented in this section. The workload is distributed among CPU cores of a cluster of computers connected via a dedicated high-speed network. The necessary information to make the CPUs to collectively solve the problem are exchanged among the CPU cores using Message Passing Interface (MPI). The basic processing unit of our current implementation is a CPU core since we use only MPI. The terms CPU core and MPI process are used interchangeably in the rest of the paper.

3.1. Domain decomposition: distributing workload among multiple computers

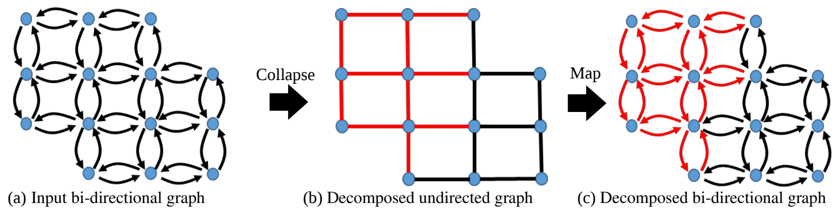

Collapse
Map
(a) Input bi-directional graph
(b) Decomposed undirected graph
(c) Decomposed bi-directional graph

Fig. 2: Main steps of generating non-overlapped partitions of a bi-directional graph.

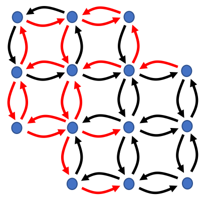


Fig. 3: Overlapped partitions

In order to distribute the computational workload of D2DTA, the bi-directional graph of the traffic network is decomposed into non-overlapping and continuous sub-networks using the graph partitioning software METIS (Karypis and Kumar (1998)). Since the amount of computations of LTM is proportional to the number of vehicles passing through each link, we use link-based partitioning scheme with a number of passing vehicles as weight. The time asynchronous nature of LTM (i.e. clocks of vehicles are not synchronized) makes it difficult to make a reasonable estimation of the amount of computations. The execution time of each link measured in previous iterations would be a better indicator of workload.

Figure 2 illustrates the main steps involved in partitioning the bi-directional graph. It is highly desirable to use non-overlapped partitions since it enables efficient traffic flow simulations within each MPI process, simple and efficient management of message passing among MPI processes, and attaining high parallel scalability by overlapping communication and computation (i.e. communication hiding). In order to obtained non-overlapped partitions, first the bi-directional graph is collapsed into an undirected graph; the sum of the number of vehicles in each pair of up and down links are assigned to the corresponding link of the undirected graph as the estimator of the amount of computations. Partitioning the undirected graph with METIS and mapping the results back to the original bi-directional graph, we can obtain desired non-overlapped partitions (see Fig. 2(b) and Fig. 2(c)). Direct partitioning the bi-directional graph with METIS produces overlapped partitions (i.e. up and down links between some pairs of nodes belong to different MPI processes) as shown in Fig. 3. Such overlapped partitions significantly lower both the computational efficiency and parallel scalability, and complicate the management of MPI communications. Each of the partitions is assigned to a separate MPI process so that computation power of multiple MPI processes can be utilized to solve the problem in a shorter time.

To maintain the continuity, each MPI process should exchange state of the upstream links located along partition boundary, which deliver vehicles to neighbor partitions, to the owner MPI processes of its neighbor partitions. In order to efficiently manage communication of the state of those upstream links, we include their dummy copies along the boundary of the neighbor partitions according to the corresponding downstream links. In each partition, the dummy upstream links are grouped into a set called *to_recv_links*. Further, the upstream links along partition boundaries, whose states are to be sent to the neighbor partitions, are grouped to a set called *to_send_links*. The set *innermost_links* is formed by subtracting the *to_send_links* from the links of a partition. In traffic flow simulations using LTM, first the *to_send_links* are executed, and the messages carrying the updated states to the corresponding neighbor partitions are posted using the non-blocking MPI functions *MPI_Isend*() and *MPI_Irecv*(). Next the traffic flow of the *innermost_links* is executed, and finally the posted messages are finalized with *MPI_Waitall*() or *MPI_Testall*(). This organization of

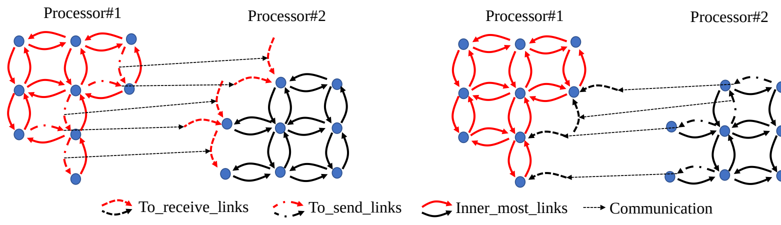
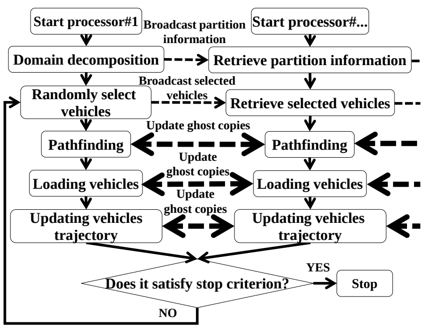
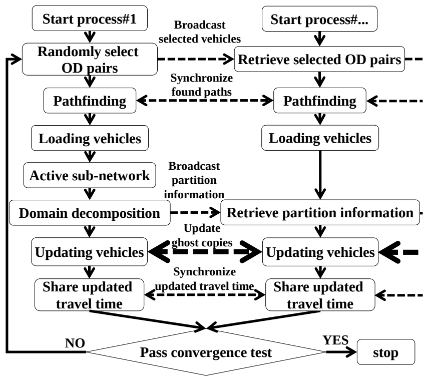


Fig. 4: Grouping of links for communication hiding and efficient communication management. At each iteration, the state of ghost copies, shown in dashed lines, are updated using MPI according to the state of the corresponding links in the neighbor partitions.


Start processor#1
Broadcast partition information
Start processor#...
Domain decomposition
Retrieve partition information
Broadcast selected vehicles
Randomly select vehicles
Retrieve selected vehicles
Update ghost copies
Pathfinding
Pathfinding
Update ghost copies
Loading vehicles
Loading vehicles
Update ghost copies
Updating vehicles trajectory
Updating vehicles trajectory
Does it satisfy stop criterion?
YES
Stop
NO

(a) With parallel path finding.



(b) With embarrassingly parallel path finding and active sub-network.

Fig. 5: Flowcharts of two implementations of parallel day-to-day traffic assignment. Dashed lines represent MPI communications; thickness and arrows indicate volume and direction of data flow, respectively.

computations and communications so that computations can be overlapped with communications makes it possible to reduce the communication overhead, thereby increasing the parallel scalability (i.e. reduction of the execution time proportional to the number of MPI processes).

3.2. *Enhancements of parallel performance*

The computation time of a perfect scalable parallel program halves when the number of CPUs doubles. Attaining high scalability is a challenging task and requires significant modifications to the data structures and serial algorithms, or it may even require the development of completely new algorithms. This section presents three strategies, apart from above presented communication hiding, to improve the computational efficiency of distributed D2DTA.

3.2.1. *Pathfinding*

Pathfinding is one of the most time-consuming components of D2DTA. We use the Dijkstra's algorithm (Dijkstra (1959)) for pathfinding. Due to its inherent serial nature, it is difficult to implement scalable distributed or shared memory parallel versions of Dijkstra's algorithm. A shared-memory parallel implementation by Jasika et al. (Jasika et al. (2012)) has attained only 10% speed up, which is fairly low. None of the distributed-memory parallel implementations (Hribar et al. (1998);Hribar et al. (2001);Chabini and Ganugapati (2002)) have reported reasonable scalability. The main difficulty in distributed parallel implementation is involvement of a large number of messages and difficulties in assigning balanced workloads to MPI processes.

We developed two versions of parallel D2DTA; one with distributed memory parallel Dijkstra's algorithm similar to Haribar et al.'s work (Hribar et al. (1998)), and the other with the serial Dijkstra's algorithm. Figure 5 illustrates details of the MPI communications involved in both the implementations. In the serial implementation, each MPI process keeps a copy of the whole network and travel time information of each link and calculates paths for a given

(a) Complete traffic network. (b) The active links during the early iterations.

Fig. 6: Decomposed Nagoya network for 4 MPI processors (Colors indicate partitions)

subset of vehicles. We assigned an equal number of vehicles to each MPI process so that each MPI process will have nearly the same computational workload. Instead of assigning the equal number of vehicles, using the spatial distance between the origin and destination of each vehicle, or previous measurements of pathfinding time will lead to better scalability. This serial implementation of pathfinding belongs to embarrassingly parallel category, since finding a path for a vehicle does not require any MPI communications.

The main disadvantage of the embarrassingly parallel approach is the need to keep of copy of complete network in each MPI process. This is not a major problem since modern computing nodes have a large memory. Though the serial implementation requires each MPI process to exchange the travel time data of the previous iterations and the paths found with the rest of MPI Processes, this communication overhead is orders of magnitudes smaller than that is required by the distributed memory parallel pathfinding. Further, synchronizing the paths among all the MPI processes make it possible to completely eliminate the large number of communications involved in vehicle loading step of parallel pathfinding. This large reduction of communication overhead and better load balance make the embarrassingly parallel approach to drastically improve the scalability of pathfinding and vehicle loading steps.

3.2.2. *Processing only the active sub-network*

Significantly small number of links are active (i.e. vehicles passes through) at the early stages of D2DTA since D2DTA algorithms gradually introduce vehicles to the simulation. Figure 6(a) and Fig. 6(b) show the full Nagoya network and active links in an early iteration of D2DTA. Therefore, it is logical to partition and simulate only the active sub-network, instead of simulating the whole network in the early iterations. Even though the inactive links do not incur any computational overhead, processing the whole network requires data of each link to be brought to CPU from main memory, which is a significant time consuming operation. Further, as mentioned in section 3.1, we use the number of vehicles in each link as the weight in partitioning. The presence of links with zero vehicles leads to seriously low-quality partitions with high workload imbalance; note that the links with zero vehicles also consume significant time in loading data from main memory.

If only the active sub-network is simulated, it will not only eliminate time wasted on unproductive loading data of inactive links from main memory but also generate well load-balanced partitions leading to higher parallel efficiency. As it will be demonstrated in the next section, there is a considerable gain in simulating only the active sub-network. The active links can be easily identified right after the vehicle loading stage. Figure 5(b) illustrates the flowchart of the developed parallel D2DTA algorithm with active sub-network.

3.2.3. *Processing links in the direction of vehicle flow*

The number of iterations requires for LTM based traffic simulator, which we use to simulate traffic flow, depends on the execution order of the links. As an example, consider a unidirectional traffic flow along a single road consisting of N number of links. LTM based traffic simulator involves only one iteration if the links are processed from origin to destination. However, if the links are processed in the destination to origin direction, it requires N number of iterations. As this simple example demonstrates, the computation time can be reduced by processing the links from upstream to downstream direction. Though it is straightforward to identify in this simple example, no such ideal execution order which complete traffic flow simulation in one iteration exists for web-like real traffic networks. Although an ideal link processing order does not exist, processing the links in the origins to destinations directions can significantly reduce the number of iterations in simulating traffic flow of real networks. Identification of near-optimal link execution order for either the whole network or each partition is a challenging graph optimization problem.

Instead of solving an optimization problem, we found a very effective links execution order which is straightforward to find in a short time. As explained earlier, we classify the links in each partition to three groups. Vehicles enter and leave a partition from *to_receive_links* and *to_send_links*, respectively. Starting from the *to_receive_links*, we traverse the network in a breadth-first order, tagging immediate downstream links of *to_receive_links* as *tier-1*, their immediate downstream as *tier-2* and so on until *to_send_links* are reached (see Fig. 10). Each link is traversed only once during this process. While this tagging process takes a short time, processing the links in the created tier-lists (i.e. *to_receive_links*, *tier-1*, *tier-2*, . . ., *to_send_links*) significantly reduces the required number of LTM iterations. As demonstrated in the next section, it can produce around 30% reduction of runtime compared to the execution of links in a random order (e.g. in the ascending order of link ID's).

4. Numerical experiments

In this section, we demonstrate the effectiveness of the parallel performance improving strategies presented in the previous section. We used the Nagoya network consisting of 152,464 links and 37,511 nodes as shown in Fig. 6(a), for the simulations presented here. To evaluate the parallel efficiency, we used speed-up factor with base 2. The speed-up factor is defined as T_2/T_N. T_2 is the execution time using 2 processors. T_N is the execution time using N processors.

4.1. Distributed-memory parallel pathfinding versus embarrassingly parallel pathfinding

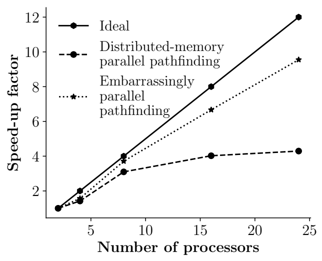


Fig. 7: Scalability of the two pathfinding algorithms.

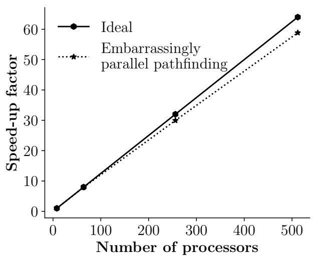


Fig. 8: Scalability of embarrassingly parallel pathfinding; a large-scale problem.

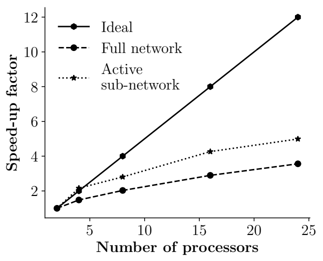


Fig. 9: Scalability improvement brought by processing active sub-network.

Two experiments are conducted to evaluate advantages of the embarrassingly parallel pathfinding, which is explained in section 3.2.1, comparing with the distributed-memory parallel Dijkstra's algorithm by Hribar et al. (1998). In the first experiment, 5,000 vehicles with random origin-destination pairs (OD pairs) are simulated in a workstation consisting of two Intel Xeon CPUs (E5-2697 v2 @ 2.70 GHz) and 256 GB memory. As shown in Fig. 7, the embarrassingly parallel algorithm produces significant improvement in scalability. Most importantly, as shown in Table 1, there is nearly two orders of magnitudes reduction of pathfinding time. In the second experiment, scalability of the embarrassingly parallel pathfinding is tested with 500,000 vehicles in the K-computer (AICS, Kobe, Japan). According to Fig. 8, near ideal scalability can be attained with this embarrassingly parallel approach. These tests demonstrate that the embarrassingly parallel pathfinding does not only have higher scalability but also requires significantly low computation time as shown in Table 2.

4.2. Processing the full network versus the active sub-network

In this section, the effectiveness of processing only the active sub-network is compared with the full network. Randomly selected 200,000 vehicles are used. The full road network (Fig. 6(a)) consists of 152,464 links and 37,511 nodes, while the active sub-network (Fig. 6(b)) consists of 3,943 links and 2,020 nodes.

According to Fig. 9, processing only the active sub-network produces only a slight scalability improvement. However, Table. 2 shows that simulating traffic flow on the active sub-network is around 30 times faster than that with the full network. This indicates that traversing through inactive links, CPUs waste a significant amount of time unproductively accessing main memory.

MPI processes	T_1 (s)	T_2 (s)	T_1/T_2
4	6,429.58	25.24	254.764
8	2,949.18	10.68	276.14
16	2,269.82	5.94312	381.924
24	2,127.69	4.15	512.969

Table 1: Execution time of distributed-memory (T_1), and embarrassingly parallel (T_2) pathfindings.

MPI processes	T_1 (s)	T_2 (s)	T_1/T_2
4	61,225.2	2,171.71	28.1921
8	44,702.7	1,671.18	26.75
16	31299.2	1,099.71	28.46
24	25452.2	939.583	27.09

Table 2: Execution time of updating vehicle trajectory with full network, T_1, and active sub-network, T_2.

MPI processes	T_1 (s)
8	9,135.34
64	6,429.58
256	2,949.18
512	2,269.82

Table 3: Execution time of embarrassingly parallel pathfinding for large scale problem, T_1.

MPI processes	T_1 (s)	T_2 (s)	$((T_1 - T_2) \times 100)/T_1$
2	4,687.23	2,769.81	40.9%
4	2,171.71	1,422.4	34.5%
8	1,671.18	1,175.33	29.7%
16	1,099.71	903.52	17.8%
24	939.58	682.5	27.3%

Table 4: Time required for traffic flow of one million vehicles with unordered processing of links, T_1, and tier-ordered processing of links, T_2.

4.3. *Processing links in the direction of vehicle flow*

In order to study the effectiveness of processing links in traffic flow direction, which was presented in section 3.2.3, two sets of traffic flow simulations were conducted with randomly selected 1,000,000 vehicles. The links were updated in the ascending order of their ID's (i.e. a random order) in the first set of simulations, while, in the second set, the links were updated according to the tier-lists presented in section 3.2.3. Only the active links of the network are considered in these simulations, and time to complete traffic flow with LTM is measured. According to Fig. 11, processing the links in the flow direction based tier list and random order have identical poor scalability.

Though tier-ordered list produces no improvement in scalability, it significantly reduces the number of iterations required for simulating the traffic flow. As shown in Fig. 12, the flow rate (i.e. the number of updated vehicles during an iteration) is significantly high in the tier-ordered processing of links. This high vehicle flow rate reduces the required number of iterations for traffic flow simulation to 28, while the unordered processing of links requires 70 iterations. Further, Table 4 shows that tier-ordered processing of links brings about 30% reduction of time required for simulating the traffic flow with LTM.

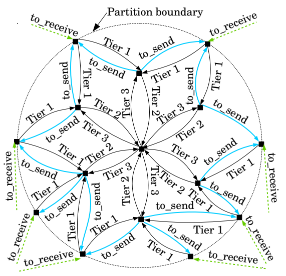


Fig. 10: Tier ordering of links according to vehicle flow directions.

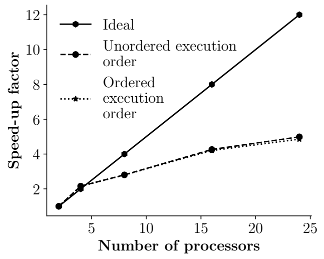


Fig. 11: Scalability of traffic flow simulations with tier-ordered, and unordered processing of links.

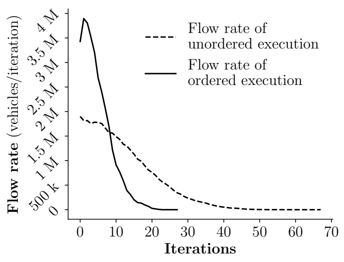


Fig. 12: Vehicle flow rates with tier-ordered, and unordered processing of links.

5. Concluding remarks

In this paper, we presented a distributed memory parallel implementation of day-to-day traffic assignment algorithm (D2DTA) which is being developed with the aim of finding near optimal traffic assignment for large networks. It is demonstrated that embarrassingly parallel pathfinding does not only substantially improve pathfinding scalability, but it is also several orders of magnitudes faster compared to parallel pathfinding. The use of only the active sub-network in early iterations greatly reduced the computational time of LTM based traffic flow simulation; nearly 20 times faster than using the full network. Further, we presented a link execution sequence which significantly reduces the number of iterations required to complete LTM based traffic flow simulations; it can reduce the traffic flow simulation time around 30%.

The remaining bottleneck is the scalability of traffic flow simulator. There is a big imbalance of execution time of each MPI processor in each iteration. The imbalance of execution time is caused by the difficulty of assigning an equal work load to each MPI processor. Therefore, in the future, the dynamic load balancer technique could be applied. This technique is promising for reducing the imbalance of runtime.

Acknowledgements

Most of the results are obtained using K computer at the RIKEN Advanced Institute for Computational Science (hp150283).

References

Attanasi, A., Silvestri, E., Meschini, P., Gentile, G., 2015. Real world applications using parallel computing techniques in dynamic traffic assignment and shortest path search, in: 2015 IEEE 18th International Conference on Intelligent Transportation Systems, pp. 316–321. doi:10.1109/ITSC.2015.61.

Chabini, I., Ganugapati, S., 2002. Parallel algorithms for dynamic shortest path problems. International Transactions in Operational Research 9, 279–302. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1475-3995.00356, doi:10.1111/1475-3995.00356, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/1475-3995.00356.

Chabini, I., Jiang, H., Macneille, P., Miller, R., 2003. Parallel implementations of dynamic traffic assignment models, in: Systems, Man and Cybernetics, 2003. IEEE International Conference on, pp. 1246–1252 vol.2. doi:10.1109/ICSMC.2003.1244582.

Chang, S.E., Nojima, N., 2001. Measuring post-disaster transportation system performance: the 1995 kobe earthquake in comparative perspective. Transportation Research Part A: Policy and Practice 35, 475 – 494. URL: http://www.sciencedirect.com/science/article/pii/S0965856400000033, doi:https://doi.org/10.1016/S0965-8564(00)00003-3.

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numer. Math. 1, 269–271. URL: http://dx.doi.org/10.1007/BF01386390, doi:10.1007/BF01386390.

Hribar, M.R., Taylor, V.E., Boyce, D.E., 1998. Termination detection for parallel shortest path algorithms. Journal of Parallel and Distributed Computing 55, 153 – 165. URL: http://www.sciencedirect.com/science/article/pii/S0743731598915027, doi:https://doi.org/10.1006/jpdc.1998.1502.

Hribar, M.R., Taylor, V.E., Boyce, D.E., 2001. Implementing parallel shortest path for parallel transportation applications. Parallel Computing 27, 1537 – 1568. URL: http://www.sciencedirect.com/science/article/pii/S0167819101001053, doi:https://doi.org/10.1016/S0167-8191(01)00105-3. applications of parallel computing in transportation.

Jasika, N., Alispahic, N., Elma, A., Ilvana, K., Elma, L., Nosovic, N., 2012. Dijkstra's shortest path algorithm serial and parallel execution performance analysis, in: 2012 Proceedings of the 35th International Convention MIPRO, pp. 1811–1815.

Karypis, G., Kumar, V., 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. 20, 359–392. URL: http://dx.doi.org/10.1137/S1064827595287997, doi:10.1137/S1064827595287997.

O'Cearbhaill, E.A., O'Mahony, M., 2005. Parallel implementation of a transportation network model. Journal of Parallel and Distributed Computing 65, 1 – 14. URL: http://www.sciencedirect.com/science/article/pii/S0743731504001303, doi:https://doi.org/10.1016/j.jpdc.2004.07.003.

Shen, Z.J.M., Pannala, J., Rai, R., Tsoi, T.S., 2008. Modeling transportation networks during disruptions and emergency evacuations. UC Berkeley: University of California Transportation Center URL: https://escholarship.org/uc/item/1257t9zn.

Smith, M.J., Wisten, M.B., 1995. A continuous day-to-day traffic assignment model and the existence of a continuous dynamic user equilibrium. Annals of Operations Research 60, 59–79. URL: https://doi.org/10.1007/BF02031940, doi:10.1007/BF02031940.

Thulasidasan, S., Eidenbenz, S., 2009. Accelerating traffic microsimulations: A parallel discrete-event queue-based approach for speed and scale, in: Proceedings of the 2009 Winter Simulation Conference (WSC), pp. 2457–2466. doi:10.1109/WSC.2009.5429673.

NOTE: This and following pages only contain the responses to reviewers. No contents to be published are included.

Message from an author to reviewers

I would like to thank all the reviewers for their valuable comments on the paper. I updated the paper according to their comments, and the rest of this document contains the details of those updates.

Reviewer 1

There are no description related to disasters after chapter 2.

As mentioned in the introduction, development of numerical tools to find near optimal traffic allocation for real-life problems is the major challenge. And, finding optimal traffic assignment after a major earthquake is one out of many possible applications of the developed system. The objective of this paper is to address this major challenge. Hence we only discuss various techniques to accelerate the process of finding near optimal solutions, not the applications. No modifications are included regarding this comment.

Although it is expected that the road condition and OD traffic demand fluctuate day by day in the event of a major earthquake disaster, the reason for assuming equilibrium is not clear.

I included more information in section 2 about the day-to-day traffic assignment and an additional reference, [Smith and Wisten(1995)], in section 2 to further explain the user equilibrium state. The definition of day-to-day traffic assignment is clearly explained in this reference.

Reviewer 2

What are the definition of Speed-up factor on the vertical axis in Figure 7, Figure 8 and Figure 9?

Detailed information on the speed-up factor is included in the section 4

Do you think that this is a big problem for the statement "this indicating that traversing through inactive links, CPUs waste a significant amount of time unproductively accessing main memory "? Is there a way to solve this problem?

The section 3.2.2 explains that this problem can waste significant computation time, and wasted time can only be eliminated by processing only the active links. hence, no modifications are made regarding this comment.

Let me give a more simpler explanation. If we process the full network shown in Fig. 6 (a), significant amount of CPU time would be wasted in reading a large amount of useless data from the main memory. It is well known that accessing main memory takes a long time; several hundreds of CPU cycles per each read. To eliminate this waste of time, we first find only the active links shown in Fig. 6 (b), and process only those active links eliminating the waste of time in reading a large amount of useless data.

Bibliography

[Smith and Wisten(1995)] Smith, M. J., Wisten, M. B., Dec 1995. A continuous day-to-day traffic assignment model and the existence of a continuous dynamic user equilibrium. Annals of Operations Research 60 (1), 59–79. URL `https://doi.org/10.1007/BF02031940`