High-Performance Computing Enhancement of Macroscopic Day-to-day Traffic Assignment

Wasuwat PETPRAKOB[1], Lalith WIJERATHNE[2], Takamasa IRYO[3], Junji URATA[4] , and Kazuki FUKUDA[5]

[1]Department of Civil Engineering, The University of Tokyo
(7-3-1, Hongo, Bunkyo, Tokyo, 113-8654, Japan)
E-mail:wasuwat@eri.u-tokyo.ac.jp
[2]Earthquake Research Institute, The University of Tokyo
(1-1-1, Yayoi, Bunkyo, Tokyo, 113-0032, Japan)
E-mail:lalith@eri.u-tokyo.ac.jp
[3]Department of Civil Engineering, Kobe University
(1-1, Rokkodai-cho, Nada, Kobe, 657-8501, Japan)
E-mail:iryo@kobe-u.ac.jp
[4]Department of Civil Engineering, Kobe University
(1-1, Rokkodai-cho, Nada, Kobe, 657-8501, Japan)
E-mail:urata@person.kobe-u.ac.jp
[5]Department of Civil Engineering, Kobe University
(1-1, Rokkodai-cho, Nada, Kobe, 657-8501, Japan)
E-mail: 163t130t@stu.kobe-u.ac.jp

A major earthquake can damage road network disrupting the passenger traffic and supply chain network for many months or years. When a commercial center like Tokyo is affected by such natural disaster, it can bring serious economic losses both in short and long term. In order to minimize the economic losses, it is vital to optimally utilize the active portion of the road network meeting the traffic demand with minimum delay. It is well known that a dynamic traffic assignment is NP-hard problem, hence it is essential to develop high-performance computing extensions to find nearly optimal solutions for this post-disaster traffic assignment. According to our literature survey, no scalable parallel implementations of a suitable traffic assignment algorithm are reported in the literature.

In this paper, we present a preliminary implementation of HPC enhanced day-to-day traffic assignment with the details of strategies to attain higher parallel scalability. According to numerical experiments, the proposed methods significantly accelerate the runtime of day-to-day traffic assignment and improve the scalability of day-to-day traffic assignment.

Key Words : *day-to-day traffic assignment, post disaster route guidance, high performance compu-ting, domain decomposition*

1. INTRODUCTION

Damages to the lifeline networks during a major earthquake can bring long lasting disruptions to manufacturing and other economic activities leading to a secondary disaster. Especially when a commercial center like Tokyo is affected by a major earthquake, the resulting secondary disaster can bring serious risk to the nation's economy and even send ripples in the global economy. It is hard to say which lifeline network plays the most critical role since the industries and other economic activities depend on these in a rather complicated manner. Road network is one of the vital element with a significant influence on the economic activities.

Depending on the severity of damages, recovery of road network after a major earthquake can take several months to years of time. As an example, it has taken 21 months to fully recover the road network after the 1995 Kobe earthquake[1)]. It is vital to find the means to optimally utilize the functioning portion of a damaged network to meet the traffic demand with minimum traffic delays so that degree of secondary economic disasters is minimized. Such post-disaster traffic assignment plans should be continuously updated according to the progress of recovery of

damaged segments.

It is well known that optimal traffic assignment problem is NP-hard problem, hence it is essential to choose a suitable algorithm and develop efficient high-performance computing extensions to find near optimal solutions for this post-disaster traffic assignment problem. There exists a number of methods to find near optimal traffic assignment[2),3),4),5)]. However, according to our literature survey, none of these methods have been applied to solve large scale problems like Tokyo probably due to the extensive computational demand. This emphasizes the need of choosing a computationally light algorithm and high-performance computing.

Out of the many methods for near optimal traffic assignment, in this study, we use the method of day-to-day traffic assignment, which mimics how people find shorter routes by changing their routes according to experiences on previous days. As one would easily guess, this method replans the route of a random subset of vehicles according to travel time information of the previous simulation, and estimate the resulting travel time by simulating the traffic after the route replanning. The above process is repeated until the total delay time reaches a certain convergence criterion. The advantage of this method is that the total computation time can be reduced by using a light weight macroscopic traffic simulator.

According to our literature survey, there are reported cases of parallel implementations of dynamic traffic assignment[3), 4)]. However, these implementations do not scale well (i.e. reduce the computation time linearly with respect to the number of CPUs used) to solve large scale problems in a shorter time by utilizing high-performance computer clusters or super computers. The current best implementation scales up to 128 CPUs, and takes around 1 hour and 30 minutes[6)] for a single day iteration. According to our estimations, it would take more than a month to find a solution for New York network. Though several months period is acceptable for regular traffic assignment problems, it is too long for post-disaster recovery problem. For practical applications in post disaster recovery, the total computation time has to be at least reduce to a week so that traffic assignment plans can be regularly updated with the progress of the recovery of damages. Hence, a better scalable HPC enhanced day-to-day traffic assignment must be implemented for solving post disaster traffic assignment problems.

In this paper, we present details of a preliminary implementation of a scalable parallel day-to-day traffic assignment method with the aim of applications in post disaster traffic assignment of large networks. Strategies for improving the scalability of each component of day-to-day traffic assignment is presented, and improvements in scalability are demonstrated with examples. Also, the remaining bottlenecks are discussed and possible remedies to improve their scalability are presented.

The rest of this paper is organized as follows. The second section presents a short introduction to the method of day-to-day traffic assignment. The third section presents HPC enhancement of day-to-day traffic assignment. The fourth section presents the numerical experiment of HPC enhancement.

2. DAY-TO-DAY TRAFFIC ASSIGNMENT

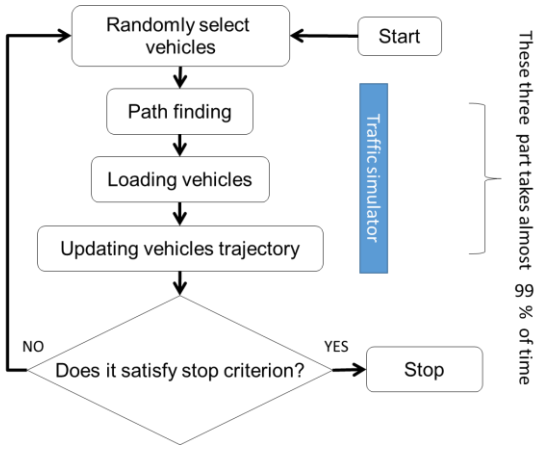


Fig.1 An overview of serial day-to-day traffic assignment.

For the sake of completeness, we present an overview of the day-to-day traffic assignment algorithm. Day-to-day traffic assignment mimics how drivers adjust their route choices to reduce their travel time or reduce their delay time based on their day-to-day travel experiences.

In this paper, a macroscopic traffic simulator is used in each iteration of day-to-day traffic simulator due to the cost-effectiveness. Each link (road segment) in macroscopic traffic simulator keeps a list of vehicles passing through the link. Each element in the links' vehicle list stores information pertaining to a vehicle such as a vehicle id, entry time to the link and departure time from the link.

The main components involved in day-to-day traffic assignment are shown in **Fig.1**. The main 5 steps involved are briefly explained below.

1. *Randomly select a set of vehicles:* If the all the vehicles are not yet included, introduce a certain percentage of vehicles from the input origin-destination pairs (OD pairs). Also, choose a random subset of the vehicles already introduced to the system.

2. *Path finding*: Find the paths with minimum travel time, based on the previous travel time estimated with step 4 of previous iterations, for the set of vehicles selected in the above step.
3. *Loading vehicles*: The randomly selected vehicles in step 1 are assigned to the vehicle lists of the links along the corresponding paths. The entry time and the departure time of newly assigned vehicles are set to infinity.
4. *Updating vehicle trajectory:* Simulate traffic flow using a suitable link delay model for handling congestions Store the travel time information of each link at suitable time interval so that travel time can be estimated in step 2 of next day iteration.
5. *Stop criterion*: Evaluate the total delay, and decide to terminate iterations or return to step 1 and continue iterations based on a suitable criterion.

The day-to-day traffic assignment's result depends on random parameters like in which order the vehicles are introduced, etc. Depending on these random parameters we may arrive at different solutions. It is best to conduct many simulations with different random parameters and choose a suitable stable solution. By the stability of the solutions can be tested in several ways. One prominent way is to give a random variation, within a practical time range, to the departure time of the vehicles and estimate the resulting traffic condition by simulating traffic flow. For this simulation for stability testing, it is better to use a microscopic traffic simulator so that most of the affecting factors can be taken into account. A criterion like the percentage increase of delay time can be used to test whether the traffic condition is not significantly affected by the random variations.

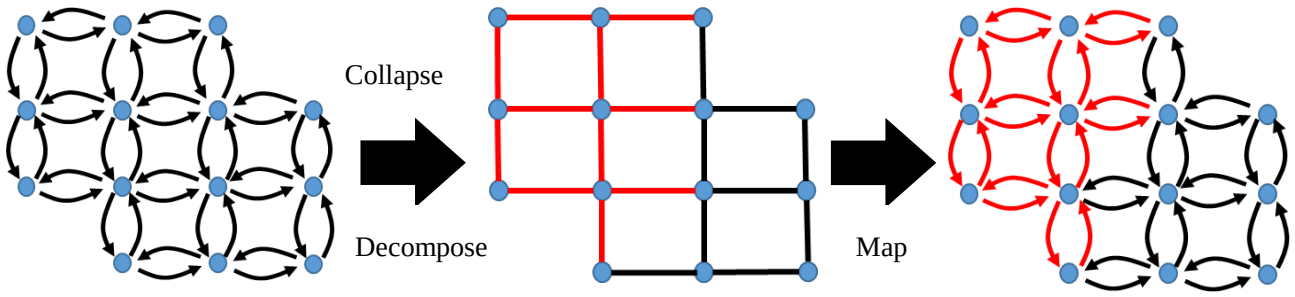
3. HPC ENHANCEMENT

As mentioned in the introduction, the post-disaster traffic assignment could not be solved unless a scalable HPC extension is developed. The post-disaster traffic assignment is one of an NP-hard problem. Therefore, we have to utilize cutting edge super computers such as K-computer to solve post-disaster traffic assignment problem for a city like Tokyo.

Most of the high-performance computing resources like computer clusters and super computers consist of a large number of computers connected with hierarchical layers of high-speed network hardware. Each computer, referred as computing nodes, may have multiple multi-core CPUs which share the memory of the corresponding computer. However, a CPU core of a given computer cannot directly access the memory of another computer, hence the name distributed memory systems. More accurately these are a hybrid system with shared memory within each node and distributed memory among computing nodes.

The communications between computer systems are made by using high-performance interconnect. A distributed-memory system is more scalable comparing to a shared-memory system.

To utilize a distributed-memory system, the work and data involved in solving a problem must be split into parts and assign to individual CPU, just as a human manager assign different works his supporting staff. Since a CPU in one node cannot access the memory of another commuting node, the work and data have to be split so that each processor can do some work independently with the data assigned to it. In order to solve the whole problem CPUs, have to exchange necessary data using the high-speed network. A library called Message Passing Interface (MPI) is widely used for this data exchange. In order to attain a good scalability (i.e. reduce the computation time linearly with respect the number of CPUs used), it is important to assign nearly equal work load to each CPU and maximize the ratio between the time for computation and data exchange.



(a) Example of road network's bidirectional graph

(b) Example of decomposed undirected graph

(c) Example of decomposed bidirectional graph.

Fig.2 Steps for partitioning a bidirectional graph

(1) Domain decomposition scheme

In order to assign each CPU core (or MPI process) an equal work load which can be independently completed with the assigned data, the network is partitioned into contiguous sub-networks. We use the graph partitioning library called METIS[7] for partitioning the network. In order to maintain the continuity, each CPU core has to exchange information incoming/outgoing vehicles to/from its subdomain with the CPU cores owning the neighboring subdomains. The partitioning scheme used in METIS minimizes the number of links intersected by the boundaries of subdomains, thereby minimizing the volume of data communicated.

There are two main partitioning scheme, node-based partitioning, and link-based partitioning. In node-based partitioning, the workload of each node can be estimated. Therefore, regions which have an equal summation of nodes' workload can be formed. Likewise, the workload of each link is approximated to be a number of passing-though vehicles in link-based partitioning. Therefore, regions which have an equal summation of links' workload can be formed.

Figure 2 illustrates the main steps involved in domain decomposition. For simplicity, we use a simple network and consider decomposing to 2 CPU cores. First, the input bi-directional graph in **Fig.2(a)** is collapsed into the undirected graph and necessary related quantities are properly aggregated. Currently, each link of the undirected graph is assigned the summation of the number of vehicles passing the corresponding bi-directional links. Assuming the computation time for each link is proportional to the number of vehicles passing through, this number of vehicles is used as a weight in partitioning with METIS so that each subdomain has an equal amount of work load. A partitioned collapsed network is shown in **Fig.2(b)**. Mapping the partition data back to the original bi-directional graph we obtain the partitioned network for the day-to-day simulations (see **Fig.2(c)**). A partitioned real traffic network is shown in **Fig.3**.; the colors represent the owner MPI rank id.

If the example road network is decomposed without collapsing, then it produces an overlapping graph network like the one shown in **Fig.4**. Such overlapping not only make the data exchange among CPU cores quite complicated but also make it difficult use some strategies to reduce or hide the data communication time among MPI processes.

If a domain decomposition guarantees to produce the continuous and non-overlapping partition, then the communication hiding technique can be employed. The communication hiding technique is to overlap the analysis of intra links with the analysis of inter links. To apply the communication hiding to the day-to-day traffic assignment, the links in decomposed partitions are categorized into 3 groups including "to_receive_link", "to_send_links", and

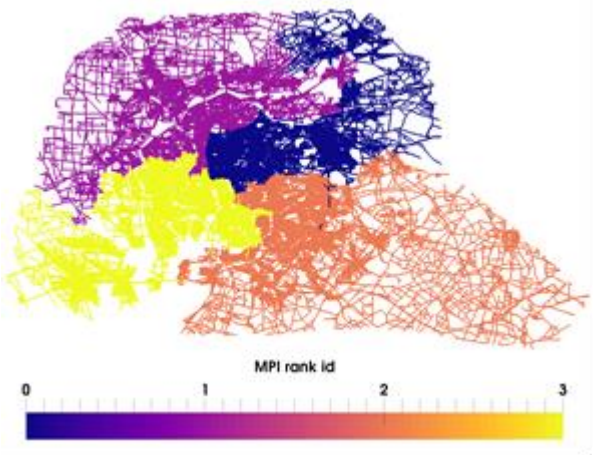


Fig.3 Example of decomposed practical road network.

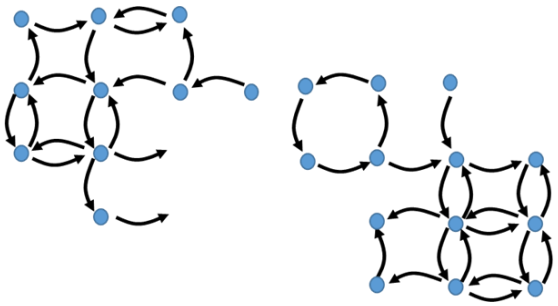


Fig.4 Example of over-lapping graphs

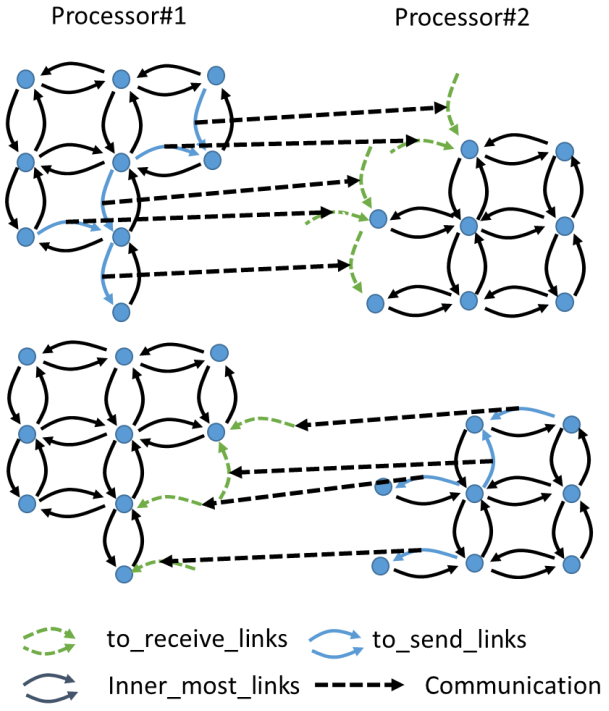


Fig.5 Communication pattern

"in-ner_most_links". The example of the communication table is illustrated in **Fig.5**. The communication hiding technique is summarized as follows:

1. The group of "to_send_links" (blue solid arrows) are analyzed and then the messages of this group are posted to neighbors. The dotted black arrows in **Fig.5** represent the messages which are posted to neighbors.
2. While we are waiting for the message from neighbors, we analyze the group of "in-ner_most_links" (black solid arrows).
3. The messages from neighbors are finalized and then the group of "to_receive_links" (dotted green arrow) are analyzed.

If we follow these steps, we do useful computations while data communication is going on, thereby completely hiding the time spent for communication. This communication hiding is a widely used standard technique in parallel computing.

As we discussed earlier, the continuity of the problem must be maintained by using ghost copies. In the day-to-day traffic assignment, the group of "to_receive_links" behaves as ghost copies. According to **Fig.5**, the group of "to_receive_links" is the same group of "to_send_links" but they belong to different processors.

(2) Parallelized day-to-day traffic assignment

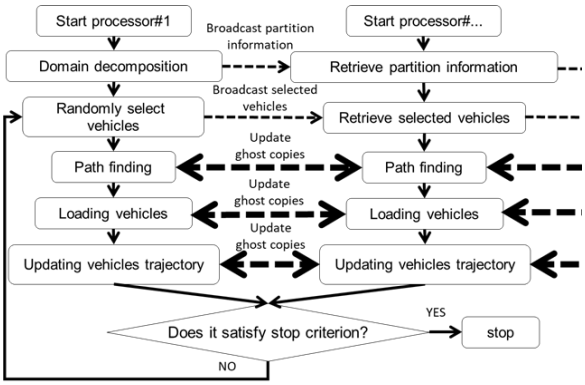


Fig.6 Flowchart of parallelized day-to-day traffic assignment.

Figure 6 illustrates the parallelized day-to-day traffic assignment. The dotted-arrows indicate the communication between processors. The unidirectional dotted-arrows represent one-way communication, and the bi-directional dotted-arrows represent two-way communication. The thick dotted arrows indicate a large number of communications of dotted-arrows indicates the number of communication messages.

(3) Path finding

We use the well-known Dijkstra's algorithm[8] for path finding. It is very difficult to parallelize[9),10),11),12)] since it is an inherently sequential algorithm. The worst case complexity of this algorithm is $O(V^2)$ where V is the number of vertices (nodes). Therefore, node-based partitioning is the best to distribute the workload to each processor.

Because of the inherent serial nature of Dijkstra's algorithm, it is difficult to implement either shared or distributed memory parallel code with a good scalability[9),10),11),12)]. According to Jasika et al. [12)] the average speed-up ratio of shared-memory parallel Dijkstra's algorithm is only 10%. Though there are several studies on distributed-memory parallel implementations of Dijkstra's algorithm[9),10),11)], none of those are scalable. The main reason for the limited scalability is the need of a large number of communications and difficulty in assigning equal work load to each MPI process.

Due to the inherent difficulties of scalable parallel implementation of Dijkstra's algorithm, we use embarrassingly parallel computation model for path finding for a large number of vehicles. In other words, instead of making the path finding the task of a vehicle parallel, we use only one MPI process for finding a path for a given vehicle. The only disadvantage of this is the need of gathering previous iterations' travel time information of the whole network to each MPI process. This demands a somewhat large amount of memory. However, it is not a serious bottleneck due to the large memory in modern compute nodes and the possibility of reducing the memory demand with some data compressing approaches. To distribute the workload for the proposed parallel path finding, each processor is assigned to find paths for an equal number of vehicles. After path finding process, each processor synchronizes the found path with every processor. Instead of calling embarrassingly parallel path finding, we call this approach synchronized-distributed-memory parallel path finding since synchronization of previous iteration's travel time and distributing the path found by a given MPI process to rest of all the MPI processes are parts of path finding for a day-to-day algorithm.

As an added benefit of above synchronization of paths, vehicles loading can be conducted without the need of any MPI communications. However, if we use a parallel implementation of path finding like the algorithm by Jasika et al.[12)], the vehicle loading involves a large number of MPI communications which can degenerate the scalability.

In the next section, the proposed path finding is compared with the parallel path finding algorithm by Jasika et al.[12)]. As it will be shown, the use of parallel

path finding involves an unknown large number of MPI communications and assigning nearly equal loads to each MPI process is quite difficult. On the other hand, the proposed path finding approach involves only 4 collective MPI commiunications and it is possible to assign nearly equal work load to each MPI process, hence producing higher scalability.

The flowchart of the day-to-day traffic assignment after applying, the synchronized-distributed-memory parallel path finding is illustrated in **Fig.9**.

(4) Active sub-network domain decomposition

According to preliminary simulations of a real road network, vehicles move along only some links (active links) in each iteration of day-to-day traffic assignment; at least during the first several daily iterations. Instead of partitioning the whole network, portioning only the active network has two advantages. The first is avoiding bad partitioning due to the use of small weights for links with no traffic; METIS does not accept zero as a weight. The second the large reduction of total computation time by eliminating the number of unnecessary memory access to main memory.

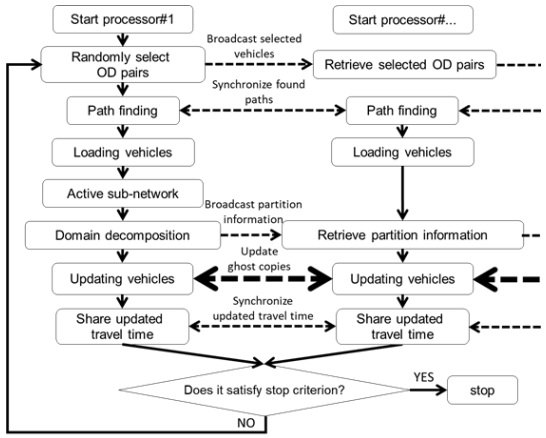


Fig.7 Flowchart of parallelized day-to-day traffic assignment with active sub-network domain decomposition.

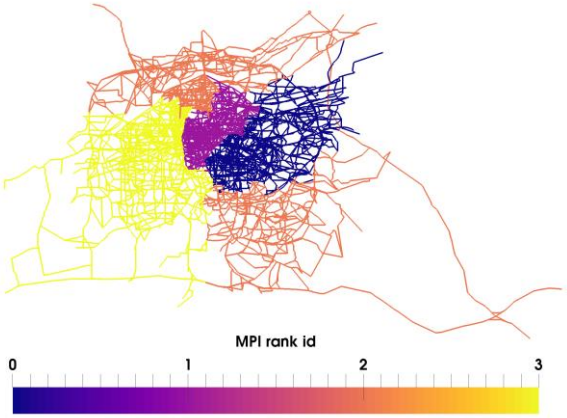


Fig.8 Example of decomposed active sub-network

Figure 7 illustrates the flowchart of parallelized day-to-day traffic assignment with the active sub-network. After loading vehicle algorithm, the inactive links can be easily identified

The active sub-network is created by picking only active links from a full network. Instead of decomposing the full network, the active sub-network is decomposed and used in updating vehicle algorithms. According to **Fig.7**, after domain decomposition scheme, only updating vehicle trajectory algorithm uses the partition information. Hence, the link-based partition is used to equally distribute the workload for accelerating the updating vehicle trajectory algorithm. The result of active sub-network domain decomposition is illustrated **Fig.8**.

The active sub-network accelerates the runtime of updating vehicle trajectory algorithm significantly because it makes the program more cache-friendly. The number of accessing unnecessary memory is eliminated. Also, the quality of partitions is improved because all partitions contain only the links that have traffic flow.

4. NUMERICAL EXPERIMENTS

In this section, we demonstrate the scalability of the parallel computing strategies mentioned in the previous section. The advantages communication hiding in scalability is not presented due to space limitations. All the numerical examples use the network shown in **Fig.3**; consists of 152,464 links and 37,511 nodes.

(1) Distributed-memory parallel path finding versus synchronized-distributed-memory parallel path finding

In order to demonstrate the advantage of using the proposed synchronized-distributed-memory parallel path finding (see section 3(3)), its scalability is compared with that of the distributed-memory parallel Dijkstra's algorithm by Hribar et al[9].

For the sake of completeness, first the distributed-memory parallel Dijkstra's algorithm[9] is briefly summarized as follows:

1. Place all vehicles' source in local subnetwork into local queue
2. Find the shortest path for all source in local queue
3. Send updated boundary node labels to neighbors
4. Receive boundary node labels from neighbors and place in queue

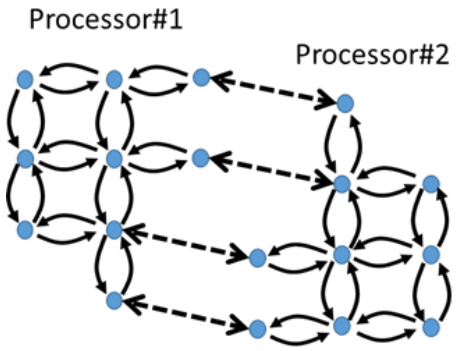


Fig.9 Communication pattern in a distributed-memory parallel path finding

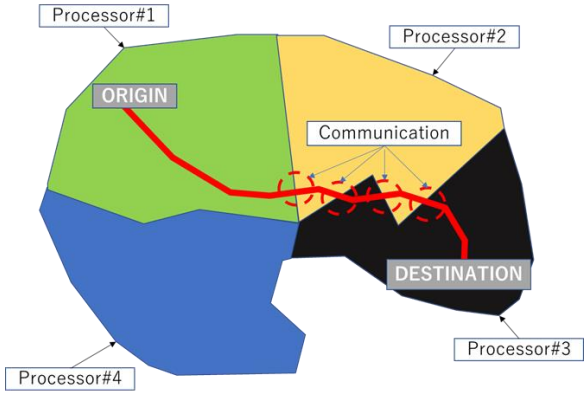


Fig.10 Communication pattern in a distributed-memory parallel path finding

5. Perform global operation for termination detection. If the algorithm is not terminated, then repeat step 2 to step 4

In this comparison, 5,000 vehicles with random OD pairs are generated. A workstation consisting two Intel Xeon CPUs (E5-2697 v2 @ 2.70GHz) with 256 GB memory is used; a total of 24 CPU cores. The scalability of distributed-memory parallel path finding is quantitatively compared with the scalability of synchronized-distributed-memory parallel path finding in **Fig.11**. As is seen the scalability of synchronized-distributed-memory is much better compared with the distributed-memory parallel path finding.

The reason for the poor scalability of a distributed-memory parallel path finding is the involvement of a large number of communication with an unpredictable pattern of destinations. The example of communication pattern of the distributed-memory parallel path finding is illustrated in **Fig.9**. The bi-directional dotted-arrow represent the communication messages. Each partition can exchange the updated node labels with their neighbors in each iteration. In some iteration, it is possible that there are inactive partitions. Therefore, the communication pattern is not predictable. To explain why it involves

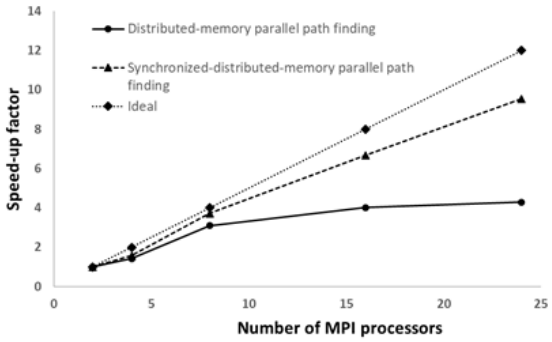


Fig.11 Scalability of parallel path finding algorithms

Table 1 Execution time of parallel path finding algorithm

#processors	Runtime of case 1 (s)	Runtime of case 2 (s)	Ratio
2	9135.34	39.64	230.5
4	6429.58	25.24	254.764
8	2949.18	10.68	276.14
16	2269.82	5.94312	381.924
24	2127.69	4.15	512.696

Case 1: distributed-memory parall path finding
Case 2: synchronized-distributed-memory parallel path finding
Ratio: $\frac{Runtime_{distributed}}{Runtime_{synchronized}}$

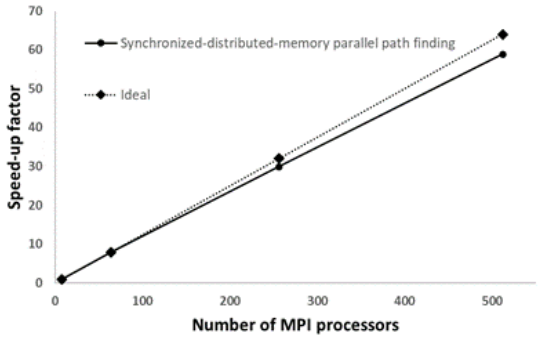


Fig.12 Scalability of synchronized-distributed-memory parallel path finding with large scale problem

Table 2 Execution time of synchronized-distributed-memory parallel path finding with large scale problem

#processors	Runtime of synchronized-distributed-memory path finding (s)	Speed-up
8	2334.6	1
64	294.5	7.93
256	77.94	29.95
512	39.6687	58.85

an unpredictable number of communications, refer **Fig.10**. Here, each partition is represented by different colors, and we consider only one vehicle with the path shown in red color. In this particular case, it needs at least 5 iterations to finish, and the exact number of iterations may be larger. In the practical problem, there are more partitions and much more vehicles leading to an unpredictable number of communication. Further, the scalability of the synchronized-distributed-memory parallel path finding is studied with large scale problem on K computer. In this experiment, 500,000 vehicles with random OD pairs. The result shown in **Fig.12** indicates that the scalability almost reaches the ideal case. The wall-clock execution times of both experiments are depicted in **Table 1** and **Table 2**. In addition to the scalability, the synchronized-distributed-memory parallel path finding is much faster than the distributed-memory parallel path finding when the number of MPI processors is increasing. Both these significantly contribute to solving post-disaster traffic assignment problem for large networks.

(2) Full network domain decomposition scheme versus active sub-network domain decomposition scheme

The objective of this set of simulations is to demonstrate the advantages of using only active network proposed in section 3(4). In this experiment, 200,000 vehicles with random OD pairs are used. The scalability of distributed-memory parallel updating vehicle trajectory was studied with two domain decomposition schemes (full domain decomposition and active sub-network domain decomposition). The decomposed full domain is illustrated in **Fig.3** and the decomposed active sub-network domain is illustrated in **Fig.8**. The full network consists of 152,464 links and 37,511 nodes, while the active network consists of 22,404 links and 37,511 nodes.

As shown in Fig.13 the scalability of distributed-memory parallel updating trajectory algorithm is significantly higher when the active sub-network domain decomposition is used. Moreover, according to the wall-clock execution times given in Table 3 the execution time of the updating vehicle trajectory algorithm with active sub-network domain decomposition is 20 times faster than the updating vehicle trajectory algorithm with full network domain decomposition.

5. CONCLUSION AND FUTURE RESEARCH

In this paper, we presented a preliminary implementation of HPC enhancements of day-to-day traffic assignment. We propose a synchronized-distributed-memory parallel path finding scheme which significantly improves the scalability of path finding. Further, it is demonstrated that using only the active sub-network the scalability of updating trajectory algorithm is significantly improved. Both these strategies significantly reduce the execution time of day-to-day traffic assignment moving us closer to use day-to-day traffic assignment to find the user equilibrium state for large scale road networks.

In the future, the scalability of updating vehicle trajectory is planned to be further improved by using the dynamic load balancing technique. Also, a mesoscopic traffic simulator might be used instead of the macroscopic traffic simulator because it would be easier to apply sophisticated traffic models such as traffic light control model or lane changing model.

ACKNOWLEDGMENT: Most of the results are obtained using K computer at the RIKEN Advanced Institute for Computational Science

REFERENCES

1) Chang, S.E. and Nojima, N. : Measuring post-disaster transportation system performance: the 1995 Kobe earthquake in comparative perspective, Transportation Research Part A: Policy and Practice Vol. 35, Pt. 6, pp. 475-494., 2001.
2) Shen, Z. M., Pannala, J., Rai, R., and Tsoi, T. S. : Modeling Transportation Networks During Disruptions and Emergency Evacuations, University of California Transportation Center. UC Berkeley: University of California Transportation Center, 2008.
3) Chabini, I., Jiang, H., Macneille, P., and Miller, R. : Parallel implementations of Dynamic Traffic Assignment models. Systems, Man and Cybernetics, 2003. IEEE International Conference, pp. 1246-1252, 2003.
4) Attanasi, A., Silvestri, E., Meschini, P., and Gentile, G. : Real world applications using parallel computing techniques in dynamic traffic assignment and shortest path search, 2015 IEEE 18th International Conference on Intelligent Transportation Systems, pp. 316-321, 2015.
5) O'Cearbhaill E.A. and O'Mahony M. : Parallel implementation of a transportation network model. Journal of Parallel and Distributed Computing Vol. 65, pp. 1-14, 2005.
6) Thulasidasan, S. and Eidenbenz S. : Accelerating traffic microsimulations: A parallel discrete-event queue-based approach for speed and scale. Proceedings of the 2009 Winter Simulation Conference (WSC), pp. 2457-2466, 2009.
7) Karypis, G. and Kumar, V. : A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. International Conference on Parallel Processing, pp. 113-122, 1995.
8) Dijkstra, E. : A note on two problems in connexion with graphs. Numerische Mathematik 1, pp. 269-271, 1959.
9) Hribar, M.R., Taylor, V.E., and Boyce D.E. : Termination

detection for parallel shortest path algorithms, Journal of Parallel Distributed Computing Vol. 55, pp. 153–165, 1998.
10) Hribar, M.R., Taylor, V.E., Boyce, D.E. : Implementing parallel shortest path for parallel transportation applications, Parallel Computing Vol. 27 Pt. 12, pp. 1537-1568, 2001.
11) Chabini, I. and Ganugapati, S. : Parallel Algorithms for Dynamic Shortest Path Problems. International Transactions in Operational Research Vol. 9 Pt. 3, pp. 279-302, 2002.
12) Jasika, N., Alispahic, N., Elma, A., Ilvana, K., Elma., L., and Nosovic N. : Dijkstra's shortest path algorithm serial and parallel execution performance analysis. 2012 Proceedings of the 35th International Convention MIPRO, pp. 1811-1815, 2012.

(Received July 31, 2017)