

Distributed Routing and Simulation of Automated Guided Vehicles

Rong YE

Wen-Jing HSU

Voon-Yee VEE

Center for Advanced Information System (CAIS)

School of Applied Science

Nanyang Technological University

Singapore 639798

{P146859245, HSU ,PA3112812}@ntu.edu.sg

Abstract: The Automated Guided Vehicles (or AGVs for short) have become an important option in the container handling process. We present a parallel simulation system for the study of AGV routing schemes. We model the AGV system based on a *time-driven* approach and execute the model on an efficient simulation engine implemented by using Cilk, a parallel programming language developed at MIT. We will elaborate on the AGV routing algorithms and traffic control scheme and propose a deadlock-free decentralized routing scheme. The performance results of the scheme are also documented.

Keywords

decentralized routing algorithm, *time-driven*, AGV.

1 Introduction

For the past five years, the businesses associated with the port of Singapore and shipping industries have earned billions of dollars for the country. Clearly, the container ports play an essential role in the national economy and it is important to streamline the container operations through IT and automation.

To improve its quality of services, the port is also continually upgraded with newer types of facilities and equipment. For port design and analysis, it is extremely useful to develop complete models and evaluate different options through comprehensive simulations. Such simulation system will be used to evaluate the container operation schemes in terms of efficiency, cost, and resource utilization.

In our study, the container operations are segregated into the following aspects for managing the complexities of the processes at the port.

- Scheduling of the port operations. This involves the decision of transferring containers from the vessels to the stacking areas and vice versa, from the stacking areas to other stacking areas and from vessels to vessels.
- The allocation of the stacking areas, the selection of the scheduling policy of container transfer, and the selection of the vehicles for routing the containers.
- Various traffic control schemes.

In this paper, we will concentrate on the second and the third aspects. The statistics collected in our simulation would give useful information to both the route layout designers and the AGV routing algorithm designers. Therefore, it should be clear that comprehensive simulation mod-

els for various port scenarios can facilitate performance analysis, study of system behavior, optimization and improvement of all aspects of the system.

1.1 Organization of the Paper

- Section 2 briefly introduces the time-driven simulation engine in our system.
- Section 3 presents how we model our time-driven AGV simulation application based on the engine.
- Section 4 describes a decentralized deadlock-free AGV routing algorithm and traffic control scheme; the performance result obtained is also summarized.
- Section 5 summarizes our work and findings.

2 A Parallel Simulation Engine

Our simulation engine is developed by using an algorithmic, multi-threaded parallel language, Cilk, developed at MIT [1,7,8]. The Cilk model is able to provide the programmer an algorithmic model of performance guarantee [2,3,1,5].

A simulation model can be viewed as a representation of the physical system under simulation. It can be classified into the continuous-state model and the discrete-state model [4]. The discrete simulation can further be classified to *time-driven* simulation and *event-driven* simulation according to how the simulation time advances. Because the *event-driven* approach is more suitable for dealing with highly asynchronous scenarios, we adopt the *time-driven* one here which is proven later more efficient for our purpose.

In the *time-driven discrete simulation* (sometimes also referred to as the unit time approach), the simulation time is incremented by a constant time step Δ . The time step Δ is largely application specific. Maintaining the accuracy of the simulation generally requires a smaller Δ , while running the simulation efficiently requires a larger value of Δ . The algorithm we adopt for it comprises a number of cycles where all components synchronize at the transition of cycles. In the algorithm, the n -th cycle corresponds to the n -th time step of the simulation. Within each cycle, the states of all components during the time interval will be simulated.

In summary, this algorithm is synchronous in the sense that it comprises a number of cycles where all processors need to synchronize at the transition of cycles.

3 Modeling AGV System

As noted, a *time-driven* discrete simulation progresses in a stepped fashion. A constant size of the time step Δ is properly chosen here to be a small quantity to advance the simulation. In the 1st time step, the behavior of each component within the time interval $[0, \Delta]$ will be simulated; in general, the behavior of each component within the time interval $[(n-1)\Delta, n\Delta]$ will be simulated in the n -th step.

In a discrete simulation system, the simulation model is decomposed into a number of submodels or components (in space domain). Each component is assigned a logical process (LP), where several LPs may be run on the same processor.

In our AGV simulation system:

- each AGV corresponds to an LP; and
- the set of tiles is a shared resource accessible by all AGVs.

In system, the tracks on which the AGVs travel are partitioned into a number of “tiles”. The size of the tiles corresponds to the range of sensors attached to the AGV in the real-life scenario. Only one AGV can occupy a tile at any instant of time. Within each time step, an AGV (i.e., an LP) locates the next adjacent tile and attempts to move into it. Conflicts arising from more than one AGV attempting to move into the same tile are resolved by enforcing exclusive access to the tile resource. Fig. 1 illustrates the actions taken by each AGV within each time step.

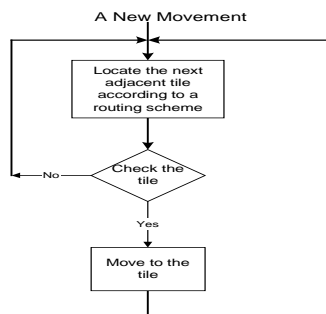


Fig. 1 Operations performed by each AGV with time-driven discrete simulation

4 AGV Routing and Traffic Control Schemes

By nature, AGV systems are concurrent (parallel and distributed) systems. Therefore, we will address certain distributed routing issues of AGVs.

Two types of AGV routing are generally identified: centralized routing algorithms and decentralized routing algorithms [6,11,9,13]. They both have their merits and demerits.

Centralized AGV Routing Algorithms

The Centralized routing algorithms are usually adopted in systems with centralized control mechanisms. A centralized routing algorithm can compute the route for an AGV when it is dispatched to serve a new pickup/drop-off job. In this case, a vehicle simply follows the routing instructions given by the central controller without making a decision itself at any points such as junctions.

Decentralized AGV Routing Algorithms

In a decentralized routing algorithm, every vehicle makes the routing decision by itself. After the central controller assigns a pickup/drop-off job to a vehicle, it has to route itself from its current position to an origin to pick up a box and then to a destination to drop off the box. Since vehicles do not have the information about the overall traffic situation, sometimes congestion or even deadlocks may occur. Therefore, deadlock detection mechanisms are required for the system and there must be solutions when deadlocks do occur. Besides, the route traveled by a vehicle may not be an optimal one, e.g. shortest distance or shortest-time path.

The centralized scheme has many advantages: Because the controller has a global view of the traffic conditions, efficient collision-prevention, congestion-prevention and deadlock-prevention algorithms can be formulated. Using continuous communications between the central controller and the AGV, the central controller can monitor the status of AGVs at all times and provide detailed error messages and statistics for routing decision.

However, despite all the potential benefits, the centralized scheme generally suffers from the high computation/communication load on the controller. As a result, the computational cost and response time will increase greatly as the size of the path network and/or the vehicle fleet increases. Another disadvantage of the centralized scheme is the high vulnerability from a single point of failure. Once a route decision is made and a vehicle is on its way, if for some reason it fails to meet its scheduled arrivals or departures, then the routing plan becomes invalidated. In addition, the system may be less fault-tolerant, because the entire system breaks down if the host computer fails.

4.1 A Decentralized Algorithm

In our system, a simple decentralized routing algorithm is adopted. That is, on a mesh layout, the AGV will monotonically decrease the distance from its current position to destination. At the junction of pathways, the AGV always chooses to move along the vertical direction first. In other words, it is a “Shortest Path, Row-First” algorithm. Lacking global information, the algorithm may lead to traffic congestion and even deadlocks in the system. Therefore, we will use auxiliary mechanisms e.g. traffic control schemes to help reduce congestion and resolve deadlocks. As mentioned earlier, those control schemes are the key components in the system, and they should be decentralized or at least partially decentralized.

4.1.1 Congestion Control and Deadlock Prevention

The algorithm described earlier takes only the topology into account but not the traffic load. When too many AGVs are present in an area and there are insufficient tiles to hold all of them, congestion will arise. Adding more tiles may help up to a point. However, the solution is not scalable as the space is normally a premium at port.

Here, we apply the *admission control idea* from the congestion control in computer networks [10]. The whole topology is divided into a set of zones. We also define a threshold $ThresholdZone_X$ for each zone according to its

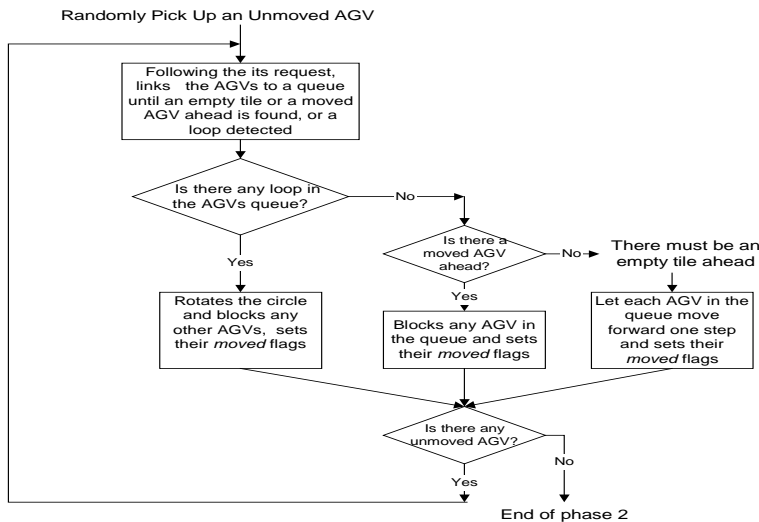


Fig. 2 Flow chart of the Centralized Algorithm

capacity and use a variable $NumofAGV_{Zone_X}$ to trace the current number of AGVs in a zone. That information is available to each zone controller. If a cross-zone movement is involved, meaning an AGV attempts to move from one zone to another zone (for example, from ZoneA to ZoneB), it must obey the following three rules:

1. If $NumofAGV_{Zone_B} \leq Threshold_{Zone_B}$, it is free to enter $Zone_B$;
2. If $NumofAGV_{Zone_B} > Threshold_{Zone_B}$, while $NumofAGV_{Zone_A} / Threshold_{Zone_A} \geq NumofAGV_{Zone_B} / Threshold_{Zone_B}$, it is still free for AGV moving from $Zone_A$ to $Zone_B$;
3. Otherwise this AGV in $Zone_A$ is forbidden from entering $Zone_B$.

By the rules, the number of AGVs permitted in a given zone at a time is limited and the traffic of a given zone will not exceed a controlled level. As a result, the AGVs will be distributed more evenly on the overall layout. The probability of deadlock within the zone is therefore greatly reduced. The scheme used in the zoning and the threshold value of the number of AGVs within each zone are two important control parameters in this technique. Because junctions are the most likely places where deadlocks occur, it is intuitive to divide the map with regards to junctions. Specifically, each zone contains one junction, with the junction centered in each zone. The following relation is used to determine the threshold number of AGVs per zone:

$$Threshold_{Zone_X} = f(Coef_{Zone_X}, NumofTile_{Zone_X}),$$

Where $Coef_{Zone_X}$ represents the congestion level of a given zone. It depends on the general traffic of the junction in the zone; $NumofTile_{Zone_X}$ denotes the number of tiles (spaces) within the zone.

4.1.2 Traffic Conduction and Deadlock Resolution

Centralized deadlock resolution algorithm

The zone control scheme by itself is not adequate to prevent the deadlock from occurring. The two figures **Error! Reference source not found.** show two cases of

cyclic deadlocks in our system where all lanes are unidirectional.

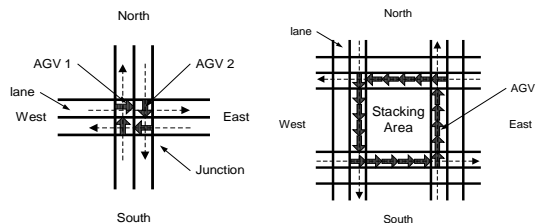


Fig. 3 Two Cycle-Deadlock Cases

For example, in Fig. 3(a), a deadlock occurs because AGV 1 intends to go east, while AGV 2 intends to move south and AGV 3 intends west and so on. The deadlock can be resolved by instructing each AGV in the cycle to move forward simultaneously. However, since each AGV has no global information about the current traffic, the deadlock occurs and AGVs are blocked by each other in a cyclic fashion. Therefore, we need to detect deadlocks and resolve them.

Here, first we will describe a centralized deadlock resolution algorithm. By analysing those two deadlock cases, we can detect them by using a simple method. Specifically, our work is to look for cycles in a directed graph and let each AGV in the cycle advance one step. To implement this scheme, each time step is further divided into two phases. In Phase 1, each AGV will locate its next adjacent tile according to the ‘‘Shortest Path, Row-First’’ rule. After working out the result, the AGV will only send a request or inquiry to that tile instead of checking and moving into it at once (as done in our first version). In Phase 2, the central controller will check the AGVs’ requests queue and make decisions for every AGV. The flow chart Fig. 2 describes the algorithm in Phase 2.

With the above two-phase algorithm, which we named the Queue-Checking Algorithm, now we are able to deal with the first three cases in Fig. 5.

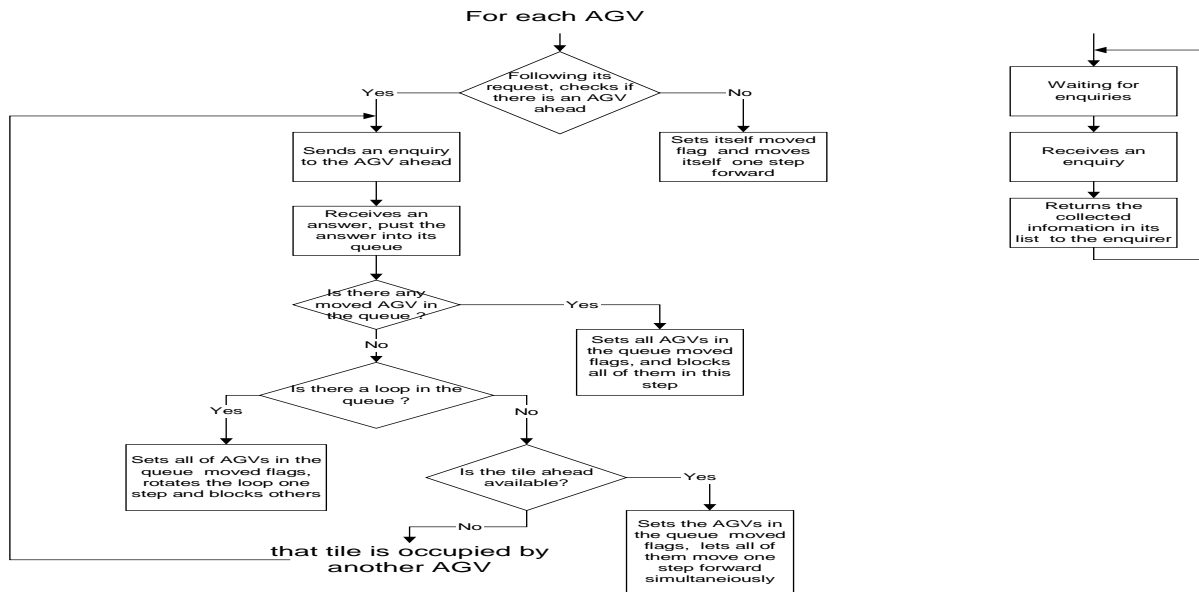


Fig. 4 Flow chart for the Decentralized algorithm

As for the fourth case, because we can only get one target tile for each AGV at a time by our routing algorithm, this situation will not occur in our system. In addition, although the first case is not an actual deadlock, it will cause unnecessary delays. We can eliminate the delays now. In short, we could ensure that we are able to cope with any cyclic deadlocks or similar cases that may occur in our system.

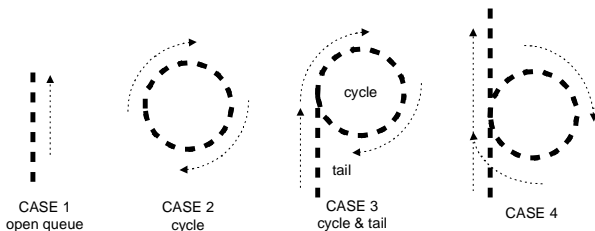


Fig. 5 Cycle Cases

Decentralized deadlock resolution algorithm

Based on this centralized algorithm, we further develop a partially decentralized version. The key idea is to allow communications among AGVs.

The partially decentralized algorithm consists of two phases as well with identical Phase 1 for the two algorithms. In Phase 2, however, each AGV will collect traffic information by communicating with the other AGVs; it also checks the AGVs queue and detects cycles based on the collected information; then it makes decisions for each of AGVs at the head of the queue. The flow chart of the algorithm is described in Fig. 4 which is similar to the one seen earlier.

At any time, an AGV will always return the latest data collected by it to its enquirers. Although there is much redundant computation involved (as each AGV's decision may be repeatedly computed by several AGVs preceding it in the queue), they also ensure a degree of fault tolerance.

4.1.3 Priority Scheme and Other Auxiliary Mechanism

In order to make sure that any AGV will eventually get to its destination, we also imposed a priority scheme and implemented an auxiliary mechanism.

1. An AGV with more blocked time will be granted a higher priority.
2. The increment of priority is randomized to reduce mutual competition recurrence.
3. An AGV with higher priority will always take precedence over a lower one when both of them try to move into a same space.
4. A tie is broken randomly when two AGVs with a same priority compete for a tile.
5. An AGV will be restored to ordinary priority upon completing a task.

Firstly, every time when we assign a task to an AGV, we will also calculate the minimum simulation steps $Steps_{min}$ needed for this AGV to move from its source to its destination. We define a value $x * Steps_{min}$ for each AGV. If after that value of simulation steps, the AGV still does not get to its destination, we will assume it has blocked in a deadlock. Then, we will promote its priority. An AGV with higher priority will prevail in the competition with the other AGVs with lower priority; furthermore, the AGV with the highest priority even can enter a zone without conforming to the zone control rules. The real worst case is when there are two AGVs that somehow gain the highest priority and compete against each other repeatedly! This scenario can be avoided by the randomization rule. A point to note is that the priority scheme is only applicable to the competition case where there are free spaces. The AGV even with the highest priority still cannot move into a tile that is currently occupied by another.

Secondly, if an AGV is blocked for some reason at the same position for more than a certain number of simulation steps, we assume that there may be a deadlock too. Conse-

quently, we will make the AGV go along another alternative. For example, in Fig. 3(a), if AGV 1 is the one who waited the longest, then after a certain number of simulation steps, it will try to go north instead of east. For AGV 1, even though the bypass may not be the shortest way to approach the destination, at least the deadlock at the junction can now be resolved. Because each lane in our layout is unidirectional, AGV 1 can move north unless there is another deadlock at the north junction and the Lane 1 is filled with AGVs all lined up. There are at least two ways to solve this problem. One is that if one of the four AGVs can move out of the current position, the deadlock is resolved. Therefore, we can make another AGV, say, AGV 2, to bypass the junction. If, at a junction, all four AGVs' alternative routes are blocked, then deadlocks at the other junctions will have to be resolved first. Once the adjacent deadlock is resolved, this deadlock is resolved too.

However, subject to the monotonicity of the routing discipline (our routing algorithm), the bypassing rule is not applied to the AGVs with the highest priority.

4.1.4 Deadlock-Free Algorithm

First of all, we have the following three premises:

1. All lanes are unidirectional and only the tiles at junctions have two exits and two entries.
2. AGVs will keep moving if they are not blocked, unless they are loading or unloading.
3. An AGV will not stay in one zone forever. That means it will try to move out of a zone sooner or later.

These premises are reasonable assumptions for AGVs used in a container terminal. With these premises, we will prove that our scheme is deadlock-free. First, we classify deadlocks in our system into two types. The first type is called a *still deadlock* in which an AGV blocks for some reason and cannot move any more. This type of deadlock, however, will not occur with our scheme for the following two reasons. With Premise 2, if an AGV is not blocked by another AGV, it will keep moving. While even if it gets blocked, the AGV still can move forward or just be held for a bounded number of steps in any case with our (*Queue-Checking*) algorithm and priority scheme.

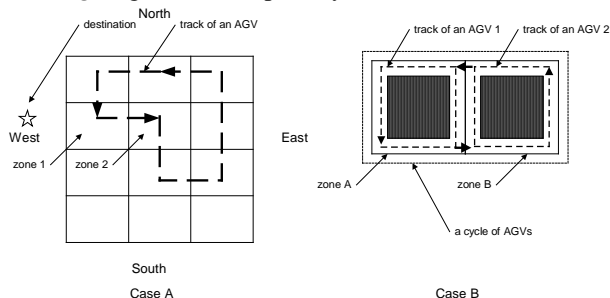


Fig. 6 Livelocks

The second type of deadlocks, called livelocks, is more intriguing. If an AGV is still moving, it cannot approach its destination anyhow as shown in Fig. 6 or other similar ways.

Our *Queue-Checking Algorithm* is able to cope with these deadlock problems. Actually, such problem can be reduced to the famous “dining Philosophers Problem”[12], with an AGV corresponding to a philosopher while a tile to a fork or a knife. The AGV holds one tile and needs to ac-

cess its next tile, which is just like a philosopher holding one fork or knife and wanting to have access of the next one. If the AGVs all hold up their tiles, all of them will suffer from the so-called “starvation”.

The difference between the two problems is that the philosopher needs both forks at a time and will release them after operations, while our AGV only needs the next tile and will release the old one as soon as it enters the new tile. Moreover, while a philosopher will eventually release all of his resources, an AGV occupies one tile at any time. Therefore, our proof should be a little different too.

As discussed earlier, if the AGVs are blocked for too long time, the priority scheme will take effect and the AGV's priorities will be promoted to a higher level. The AGV with the highest priority could run without subjecting to the congestion control rules presented earlier.

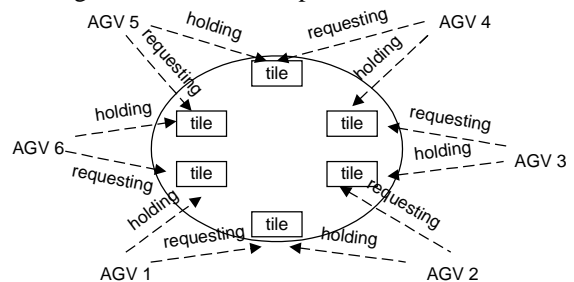


Fig. 7 AGVs' Dining Philosopher Problem

To prove that our routing scheme is deadlock-free, we classify the possible blocking scenarios into two types and show them to be all resolvable.

- Case 1: the Closed Interlock:

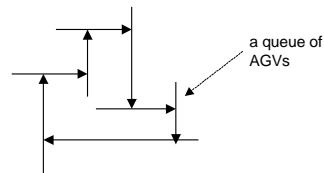


Fig. 8 Closed Interlock

In this scenario, several queues of AGVs are blocked by each other in a cyclic fashion. With our queue-checking algorithm, the cycle can be detected and hence all the AGVs in the cycle will be able to advance (disregarding their priorities). Therefore, the cycle will not hold up any AGVs within it. Therefore, by our Premise 3 and the monotonicity of the routing discipline, as long as these AGVs are keeping moving, some of them will try to move out of this cycle after a definite number of time steps. With our Priority Rule 3,4, they will successfully move out of the cycle sooner or later, so the cycle cannot persist forever.

- Case 2: the Open Interlock:

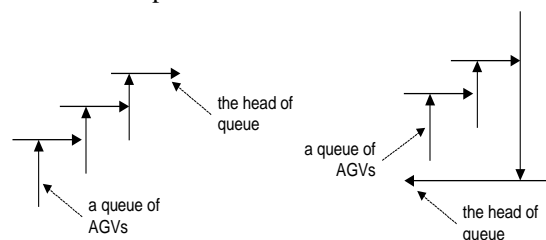


Fig. 9 Open Interlock

In this case, it should be clear that the head of the waiting queues can rightly or eventually move along intended direction. As a result, the other AGVs in the waiting queues could advance themselves too after a certain time.

In either case, none of the AGVs will be blocked forever with our queue-checking algorithm. Therefore, we conclude that our algorithm is deadlock-free.

The Fig. 10 shows the effectiveness of the traffic control scheme.

In our experiments, a task is defined as one AGV picking up a box from the ship and transferring it to the storage area or a reversed trip. From the chart, we can see, when the number of AGVs is small, there is no difference between the two schemes. But as the number of AGVs increases, it is obvious that the scheme with traffic control surpasses the one without traffic control. Fig. 10 shows traffic control scheme greatly reduces the total blocked steps in the system as the number of AGVs grows, which leads to more task completions.

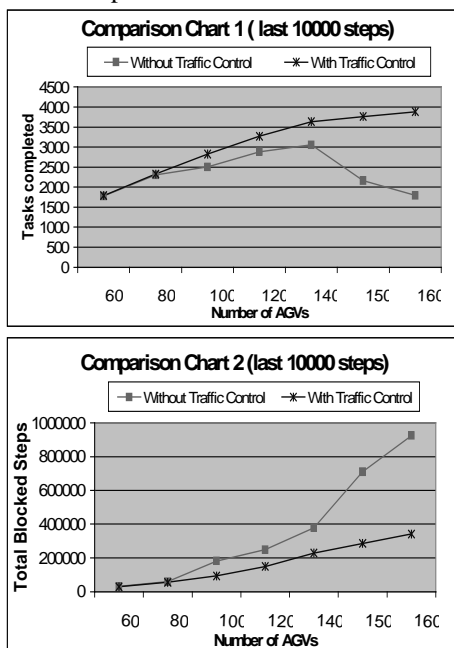


Fig. 10 Performance Comparison Charts

5 Conclusions

This project arises from the need to improve container operations by providing a higher level of services through efficient planning and management of port facilities. We have presented an efficient model for the AGV simulation system based on a simulation engine implemented by using Cilk. Given this model, we have designed and implemented a deadlock-free decentralized AGV routing algorithm and traffic control scheme. We have also proven that correctness of our algorithm. For future work, it will be most useful to incorporate AGV scheduling algorithms for performance study.

References:

[1] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and

Yuli Zhou. Cilk: an efficient multithreaded runtime system. In Proceedings of the 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pages 207--216, Honolulu, Hawaii, July 1995.

[2] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS), pages 356--368, Santa Fe, New Mexico, November 20--22, 1994.

[3] Robert D. Blumofe. Executing Multithreaded Programs Efficiently. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, September 1995.

[4] Alois Ferscha. Parallel and distributed simulation of discrete event systems. In Handbook of Parallel and Distributed Computing. McGraw-Hill, 1995.

[5] Matteo Frigo, Charles E. Leiserson, and Keith H. Randall. The implementation of the Cilk-5 multithreaded language. In 1998 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'98), Montreal, Canada, June 17--19, 1998.

[6] Chang W. Kim and J.M.A. Tanchoco. Conflict-free shortest-time bidirectional agv routing. International Journal of Production Research, 29(12):2377-2391,1991.

[7] MIT Laboratory for Computer Science. Cilk-5.2 Reference Manual, July 21, 1998. Available on the Internet from <http://supertech.lcs.mit.edu/cilk/>.

[8] Keith H. Randall. Cilk: Efficient Multithreaded Computing. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1998.

[9] J. T. L. Soh, Wen-Jing Hsu, S. Y. Huang, and A. C. Y. Ong. Decentralized routing algorithms for automated guided vehicles. In Proceedings of ACM Symposium of Applied Computing, pages 473--479, 1996.

[10] Andrew S. Tanenbaum. Computer Networks. published by Prentice Hall, Inc. ,1996

[11] F.Taghaboni-Dutta and J.M.A. Tanchoco. Comparison of dynamic routing techniques for automated guided vehicle system. International Journal of Production Research, 33(10):2653-2669, 1995.

[12] William Stallings, Operating Systems: internals and design principles, 3rded, 1998 by Prentice Hall, Inc.

[13] X. Yu and S. Y. Huang. A centralized routing algorithm for agvs in container port. In Proceedings of the 4th International Conference on Computer Integrated Manufacturing, pages 589--600, 1997.