Parallel Simulation of AGVs in Container Port Operations

Rong YE Voon-Yee VEE Wen-Jing HSU Sneha SHAH
Center for Advanced Information System (CAIS)
School of Applied Science
Nanyang Technological University
Singapore 639798
{P146859245, PA3112812, HSU, SA0750578}@ntu.edu.sg

Abstract

We describe parallel simulations of an Automated Guided Vehicle (AGV) system for the container handling at a port. The AGV system is modelled with a time-driven approach and executed on efficient simulation engines implemented by using Cilk, a multi-threaded parallel programming language developed at MIT. The speedup results of the AGV simulation over sequential versions are documented. We also present congestion control schemes of our AGV routing system.

1. Introduction

The businesses associated with the container ports and shipping industries amount to billions of dollars every year. Clearly, the container handling is an important business and it is therefore very important to promote the efficiency of the container handling process. However, with hundreds of pieces of equipment and numerous handling steps, a container port is a very complex system. To achieve a high efficiency, many aspects of operations must be optimized simultaneously to remove any bottlenecks. To manage the complexities of the processes, however, we will divide the container operations into the following aspects:

- Quayside operations: This includes the unloading process and the reversed loading process. For the unloading process, it involves the scheduling of vessels to the quayside; sequencing of the containers to be unloaded and the handling of the containers onto the vehicles.
- Transfer operations: It involves the dispatching, scheduling and routing of vehicles, the traffic rules for congestion controls, deadlock detection/resolution schemes, etc.
- Stacking operations: This involves the decision of stacking location, the sequencing of movement at the locations, e.g., the sequence to stack or un-stack a sequence of containers.

In this paper, we will concentrate on the second aspect of the container operations. Even so, the simulation of this aspect with high level of details still requires large amount of time with the sequential computers, and parallel computers are naturally acknowledged as the most promising solution to alleviate this problem. Nevertheless, the parallel hardware, by itself, will not allow us to automatically solve the problem. We still need commensurably competent parallel software to fully exploit the potentials of parallel computers.

Because of its apparent importance, there has been much active research in parallel simulations. However, to date, there has not been a commercially available parallel simulation engine that offers reliable simulation performance. Therefore, to facilitate the simulations of port scenarios, our project not only aims to develop an AGV simulation application, but also aims to build a parallel simulation engine that could offer a certain degree of performance assurance.

Organization of paper

- Section 2 briefly introduces two versions of our parallel simulation engine, i.e. the *event-driven* version and the *time-driven* version.
- Section 3 describes an approach for modeling the AGV simulation system based on the *time-driven* simulation engines and discusses the performance results.
- Section 4 presents our partly decentralized traffic control scheme and AGV routing algorithm for reducing and resolving deadlocks in the AGV running system.
- Section 5 summaries our findings and briefly discusses possible future work.

2. Parallel simulation engine

Our simulation engine is developed by using an algorithmic, multi-threaded parallel language, Cilk [1,9, 12, 13]. The Cilk model is able to provide the programmer an algorithmic model of performance

guarantee [1, 2, 3, 8]. In particular, it guarantees a near-linear speedup for programs written following the Cilk programming paradigm. The readers are referred to the literature for the features of Cilk.

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 [5]. The discrete-state simulation models can further be classified into *time-driven* simulation and *event-driven* simulation according to how the simulation time advances:

- 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. Thus a simulation comprises a number of cycles where all model objects synchronize at the transition of cycles. In this 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 *the event-driven approach*, the increment of simulation time is triggered by the next earliest occurring event. The algorithm we adopted here comprises a number of cycles where all LPs synchronize at the transition of cycles. The ***safetime*** algorithm [see, e.g. 7,10] is used to determine the events that may be processed within each iteration. Because all the events processed within each cycle are guaranteed to be safe, the algorithm is based on a synchronous conservative method. The reader is referred to [7,10] for detailed descriptions of the ***safetime*** computation.

Both algorithms are synchronous in the sense that they comprise a number of cycles where all processors synchronize at the transition of cycles.

Because of their negligence of the cache effects and the significant overheads in context switching, the existing algorithms have not been very successful in delivering good speedups. It needs extensive knowledge to fine-tune parallel simulation programs. In our engine, we employed the following innovative and original techniques [15,16,17, 18].

- **Better utilization of memory hierarchy**

  We designed and implemented a load balancing mechanism that is cache-aware with minimal overhead. The key observation is: when one processor has just processed a simulation object (a *logical process*, or LP), most information related to the LP will still be cached by this processor. By ensuring that an LP often be processed by the same processor, our cache-aware load-balancing mechanism exploits both spatial and temporal localities and makes better use of the memory hierarchy (esp. primary and secondary caches).

- **Better load balancing mechanism**

  We have designed a number of load-balancing schemes that rely on the following ideas (refer to [15] for detailed descriptions):
  - Select victims systematically from neighbors
  - Teach other thieves to select victims more smartly
  - Use preprocessing techniques to speed up selection
  - Allow fully concurrent victim selections (zero lock contention)

- **Minimize context switching overhead with persistent Cilk threads**

  Instead of spawning one Cilk thread to simulate an LP within each step, we spawn one Cilk thread for each processor throughout the entire run of simulation and replace context switching with processor synchronization. We refer to the Cilk threads thus created as *persistent Cilk threads*. They are persistent as they are spawned at the beginning of a Cilk execution, and they are not returned (and thus 'persistent') until the end of the execution of the entire program. The use of persistent Cilk threads in Cilk programs appears to be an innovation because it is a reversal of the well-advocated spawn-and-synchronize paradigm where ephemeral child/slave threads are spawned to concurrently carry out the bulk of work. This change indeed demonstrated impressive speedups in our system.

With these innovations, we built up a rather powerful parallel simulation engine called "Silk", meaning "Simulation by using Cilk". With it, we can model our AGV simulation system on parallel SMP computers.

3. Modelling AGV simulation system

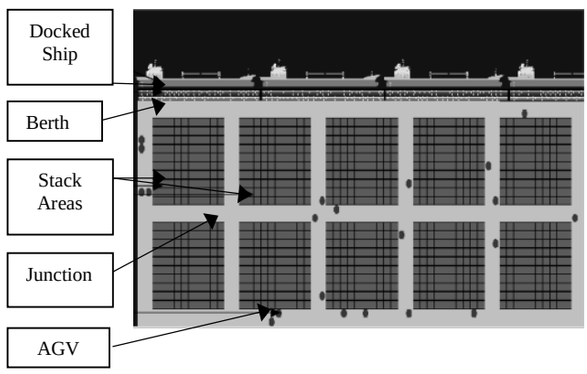

Figure 3.1 Simulation Port Layout

Fig 3.1 shows the layout in our system. Vessels will arrive at the upper side to load or discharge containers, while the AGVs (represented as small blobs) transfer the containers in between the quayside and the stacking yards.

Firstly, we developed a system based on an *event-driven* simulation engine. However the *event-driven approach* did not achieve obvious speedups. In fact, in some cases, we got speed-downs instead. The reason is that the *event-driven* simulation engine is more suitable for dealing with highly asychronous scenarios. Therefore, for our AGV system, which does not correspond exactly to such a scenario, we developed a more efficient *time-driven version*.

3.1 Time-driven version

In the *time-driven* discrete simulation, 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.

A *time-driven* discrete simulation progresses in a stepped fashion. A constant size of the time step Δ is chosen 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. The time step Δ is largely application specific and is properly chosen here to be a small quantity. Maintaining the accuracy of the simulation generally requires a smaller Δ, while running the simulation efficiently requires a larger value of Δ.

In this AGV simulation system, the tracks on which the AGVs travel are partitioned into a number of **tile**s. 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.

In this version:

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

Within each time step, an AGV (i.e., an LP) locates the adjacent tile it will move to and attempts to move into the tile. Conflicts arising from more than one AGV attempting to move into the same tile are resolved by enforcing exclusive access to the tile resource. Figure 3.2 illustrates the actions taken by each AGV within each time step.

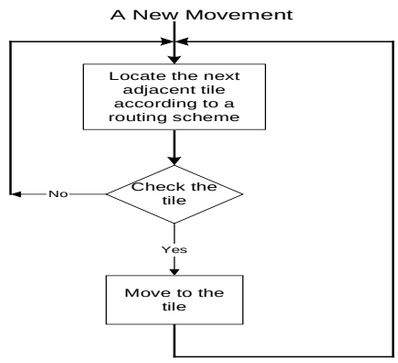

Figure 3.2 Operations performed by each AGV with *time-driven* discrete simulation

3.2 Results and discussion

Fig. 3.3 shows the speedups obtained using the *time-driven* simulation approach with and without load balancing. The same algorithms were run on different numbers of processors to obtain the speedup values. To scale beyond the limit of our 6-processor machine, this set of experiments was carried out on a 32-processor SGI Origin 2000. This parallel computer, however, is a heavily shared system, which has caused difficulty in gathering speedup figures beyond 10 processors.

Notably, with our load-balanced *time-driven* approach, we have obtained:

- **a speedup of 7.34** assuming 1,000 operations per move(using 10 processors); and
- **a speedup of 4.58** assuming 100 operations per move(using 7 processors).

We would like to note here that all the experiments were carried out during the time the system was loaded by more than 40 users with a workload of more than 10 processors fully-occupied.

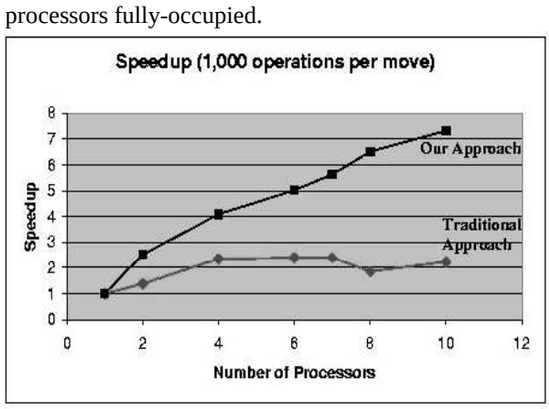

Figure 3.3 (a)

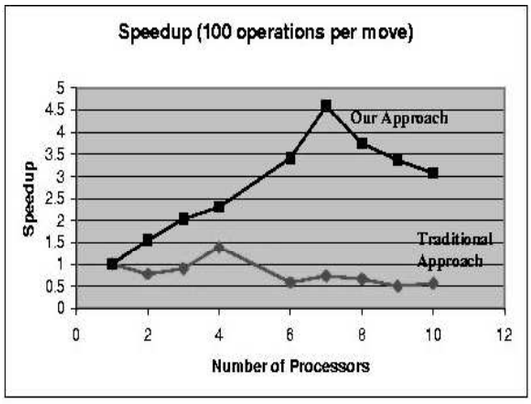

Figure 3.3(b)

The speedups obtained for AGV simulations by assuming (a) 1,000 operations per move and (b) 100 operations per move respectively. The dip of performance beyond 8 processors is caused by heavy usage of the shared system.

4. AGV routing and traffic control schemes

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

Two types of AGV routing are generally identified: *centralized AGV routing algorithm and decentralized AGV routing algorithm* [11, 14, 19].

- **Centralized AGV routing algorithms**

The system uses a central controller for scheduling and routing. In this case, the AGVs do not have a map of the routes and they just act according to the instructions of the central controller. An AGV runs from the start point to its destination by executing a sequence of instructions pre-calculated by the central controller. Alternatively, AGV receives instructions from the central controller only at critical decision point (e.g., a junction) on which path to take and when to stop or proceed.

- **Decentralized AGV routing algorithms**

In this case, the AGV has full knowledge of the routes and is able to guide itself from the source to destination, requiring only the entry of a destination. In such a system, one or more simple controllers can be used to assist controlling the traffic, preventing and resolving deadlock, and so on.

In our AGV simulations, a decentralized routing algorithm is evaluated for performance on a mesh layout. In this routing algorithm, the AGV will monotonically decrease the distance from current position to destination. At the junction of crossroads, the AGV will always choose to move along the vertical direction first. In other words, it is a "Shortest Path, Row-First" algorithm.

Congestion control

Lacking global information, a decentralized routing algorithm may lead to traffic congestion and even deadlocks in the system. Therefore, we need auxiliary mechanisms to help reduce congestion and resolve deadlocks.

- **Deadlock prevention**

We employ a zone control technique in our system. The zone control divides the whole topology into a set of zones. Only a limited number of AGVs are permitted in a given zone at a time. As a result, the traffic of a given zone will not exceed a controlled level and the AGVs will be distributed more evenly on the whole topology. The probability of deadlock within the zone is therefore greatly reduced. However, the scheme used in the zoning and the number of vehicles within each zone are two important control parameters in this technique. Because junctions are the most likely place where deadlocks occur, it is intuitive to divide the map by junctions. With our scheme, each zone contains one junction, and the junction is centered in each zone. The following relation is used to determine the maximum number of AGVs per zone:

$NUM_{AGV} = f(Coef_{zone}, NUM_{tile})$, where $Coef_{zone}$ represents the congestion level of a given zone, which depends on the general traffic of the junction in the zone; NUM_{tile} denotes the number of tiles(spaces) within the zone.

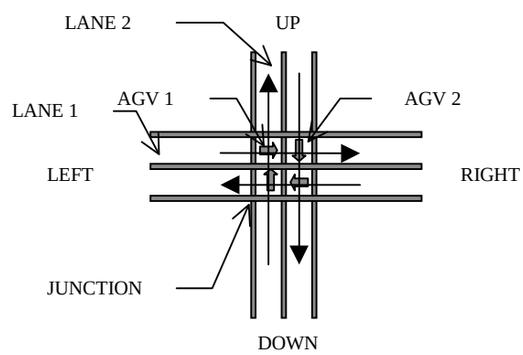

Figure. 4.1 Deadlock

- **Deadlock resolution**

Even with the zone control technique, the deadlock may still occur at a junction. Fig. 4.1 shows a typical deadlock where all lanes are unidirectional. In this scenario, a deadlock occurs because AGV 1 intends to go right, while

AGV 2 wants to move down and AGV 3 wants left and so on. Therefore, we need a mechanism to detect deadlocks and resolve them. If an AGV is blocked for some reason for more than a certain predefined number of simulation steps, we assume that there may be a deadlock. Consequently, we will make the AGV go along another way which is alternative to the original direction determined by the routing algorithm. For example, if AGV 1 is the one who waited the longest, then after a certain number of simulation steps, it will try to go up instead of right. For AGV 1, perhaps the bypass is not the shortest way to approach its destination, however, at least the deadlock at the junction is resolved. Because each lane in our layout is unidirectional, AGV 1 can move up unless there is another deadlock at the upper junction and 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 for example AGV 2 to bypass the junction. If, at a junction, all of four AGVs' alternative routes are blocked, then deadlocks at the other junctions will have be resolved first. Once the adjacent deadlock is resolved, this deadlock is resolved too. Because the total number of tiles in the overall path layout is much more than the total number of AGVs, at any time, there must be zones that have light traffic in the system. By using the "detouring scheme" described earlier, the deadlocks in the adjacent zones of these lightly-loaded zones can be resolved, which in turn will also help resolve the deadlocks in the other zones. Of course, it takes more rigorous arguments to *prove* that the system is free from long lasting deadlocks. This issue will be addressed specifically in further research work.

Simulation Results Comparison and Analysis

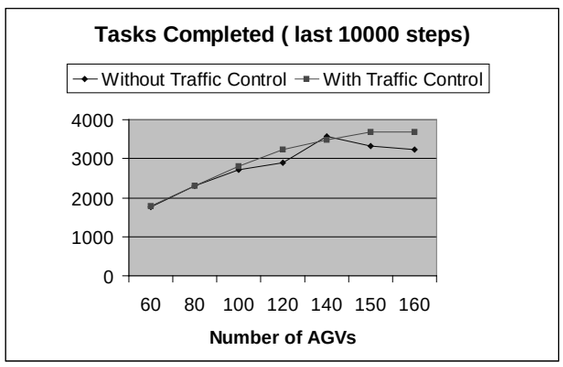

Figure 4.2 Toal Number of Completed Tasks vs the Number of AGVs

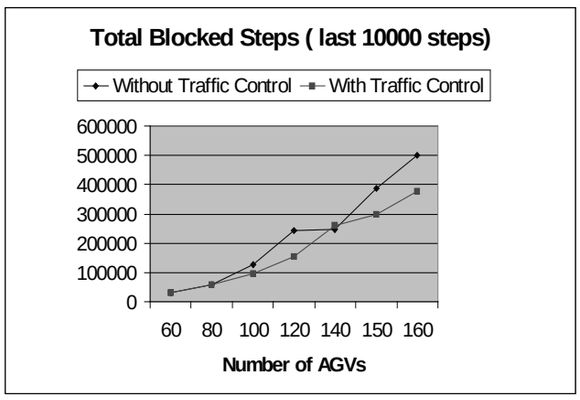

Figure 4.3 Total Number of Blocked Steps vs the Number of AGVs

One *task* is defined as one AGV picking up a box from the ship and transferring it to the storing area or a reversed movement. From the chart, we can see that, 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. 4.3 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.

5. Conclusions

This project arises from the need to improve container operations by providing the higher level of services through efficient planning and management of port facilities. Using efficient simulation engines that are implemented by using Cilk, we have presented an efficient model for the AGV simulation system. The following are recaps of our main contributions:

(1). We have designed and implemented a successful parallel simulation of the port operations resulting in nontrivial speedups.

(2). We have also developed efficient AGV routing algorithms with the help of the simulations.

This system, although still a research project and under further developments, will be a useful basis for future research and developments in this area. The completed result will help the port designer and planner to evaluate different options more comprehensively in shorter time, and thereby provide a higher-level of service while using resource more efficiently.

Acknowledgment

We wish to thank Professor Charles Leisserson of MIT for sharing the use of his Cilk parallel programming environment. We also acknowledge the Maritime and Port Authority of Singapore for its support in related projects.

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] James A. Chisman. Introduction to Simulation Modeling Using GPSS/PC. Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

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

[6] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. Introductions to Algorithms. MIT Press, 1990.

[7] Wentong Cai, Emmanuelle Letertre, and Stephen J. Turner. Dag consistent parallel simulation: a predictable and robust conservative algorithm. In Proceedings of 11th Workshop on Parallel and Distributed Simulation (PADS'97), pages 178--181, Lockenhaus, Austria, June 10--13, 1997.

[8] 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.

[9] Christopher F. Joerg. The Cilk System for Parallel Multithreaded Computing. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, January 1996.

[10] Yoke-Hean Low, Chu-Cheow Lim, Boon-Ping Gan, Sanjay Jain, Wentong Cai, Wen Jing Hsu, Shell Ying Huang and Stephen J. Turner, "Conservative Parallel Simulation for Manufacturing of Manufacturing System", 8th International Parallel Computing Workshop (PCW'98), pp. 293 - 300. September 7 – 8, 1998, Singapore.

[11] Jeong-Hoon Lee, Bum Hee Lee, Myoung Hwan Choi, Jung Duk Kim, Kwang-Taek Joo and Hyon Park, “A Real Time Traffic Control Scheme for a Multiple AGV System”, IEEE International Conference on Robotics and Automation, 1995 IEEE

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

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

[14] 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.

[15] Voon-Yee Vee and Wen-Jing Hsu. Locality-perserving mechanism for synchronous simulation algorithms. Technical report, Centre for Advanced Information Systems, Nanyang Technological University, Singapore, August 1999. Available on the Internet at http://www.cais.ntu.edu.sg:8000/.

[16] Voon-Yee Vee and Wen-Jing Hsu. Parallel discrete event simulation: a survey. Technical report, Centre for Advanced Information Systems, Nanyang Technological University, Singapore, August 1999. Available on the Internet at http:// www.cais.ntu.edu.sg:8000/.

[17] Voon-Yee Vee and Wen-Jing Hsu. A scalable and efficient storage allocator on shared-memory multiprocessors. In International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99), pages 230--235, Fremantle, Western Australia, June 23--25, 1999.

[18] Voon-Yee Vee, Rong Ye, Shah Sneha and Wen-Jing Hsu, Meeting Challenges of Container Port Operations in the Next Millennium, Gold Award, SGI-IHPC CRAYQUEST’99, Singapore, 1999.

[19] 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.