Distributed Simultaneous Task Allocation and Motion Coordination of Autonomous Vehicles Using a Parallel Computing Cluster

A.K. Kulatunga, B.T. Skinner, D.K. Liu, and H.T. Nguyen

Mechatronics and Intelligent Systems Group, Faculty of Engineering,
University of Technology, Sydney
{asela.kulatunga, brad.skinner, dkliu,
hung.nguyen}@eng.uts.edu.au

Abstract. Task allocation and motion coordination are the main factors that should be considered in the coordination of multiple autonomous vehicles in material handling systems. Presently, these factors are handled in different stages, leading to a reduction in optimality and efficiency of the overall coordination. However, if these issues are solved simultaneously we can gain near optimal results. But, the simultaneous approach contains additional algorithmic complexities which increase computation time in the simulation environment. This work aims to reduce the computation time by adopting a parallel and distributed computation strategy for Simultaneous Task Allocation and Motion Coordination (STAMC). In the simulation experiments, each cluster node executes the motion coordination algorithm for each autonomous vehicle. This arrangement enables parallel computation of the expensive STAMC algorithm. Parallel and distributed computation is performed directly within the interpretive MATLAB environment. Results show the parallel and distributed approach provides sub-linear speedup compared to a single centralised computing node.

1 Introduction

Task allocation (scheduling) and path planning (routing) are important activities which aid in the coordination of multiple autonomous vehicles in fully-automated and semi-automated material handling systems. Furthermore, these two factors are interrelated. A system which coordinates multiple autonomous vehicles effectively should address the task allocation and path planning issues. However, most of the research work focuses on these issues separately, due to the added complexity of simultaneous computation. For example scheduling and task allocation aspects have been discussed in [1-4], path planning and routing in [5, 6], deadlock detection and collision avoidance[7, 8].

However, few efforts combined these factors in their approaches [9-11]. Correa et.al., [12] have developed a hybrid approach to solve dispatching and conflict free routing of Automated Guided Vehicles (AGV) in Flexible Manufacturing Systems (FMS). Assignments and routing is considered simultaneously in their approach. However, a restriction of this work is the reduction in efficiency as the number of AGV's is increased beyond six AGV's. In addition, the online dispatching method developed by [13] uses multi-attribute dispatching rules. However, the routing is done

by considering the shortest path only and constant speeds are assumed for the AGVs. Another approach developed for dispatching and conflict-free routing of AGV's in FMS is discussed in [14]. This approach is limited to four, constant velocity AGV's. Furthermore, when a path is being utilised by one AGV, all nodes linking that path are locked and no other AGV can use the path until the assigned AGV has completed its task.

Typically path planning is followed by deadlock prevention or collision avoidance. As a consequence, optimal results acquired in the first operation (path planning) will not be the same after it is changed in order to overcome collision problems. Therefore, we can not guarantee optimal results for the both factors at the same time. Conversely, if all the factors affecting the coordination problem are considered simultaneously, then there will be a possibility of finding optimal or near optimal solutions, but it would require considerable computation time. In the STAMC approach [15] we perform path and motion planning at the task allocation stage, using the Simultaneous Path and Motion Planning algorithm (SiPaMoP) [16]. The SiPaMoP algorithm is able to search for the most efficient path by considering future deadlocks and collisions.

The STAMC algorithm is computationally intensive as it solves task allocation, path planning, deadlock and collision avoidance simultaneously. Currently, the expensive STAMC algorithm executes on a single serial computing node using MATLAB, for all autonomous vehicles in the simulation. To reduce the computation time of the STAMC algorithm, we introduce a distributed and parallel computing topology using the interpretive MATLAB environment.

The integration of the Message-Passing Interface specification into the existing MATLAB environment and STAMC algorithm code is made possible with MPITB. MPITB enables coarse-grain and out-of-loop parallelisation of the expensive SiPaMoP algorithm on distributed nodes of a Linux computing cluster.

This paper is organised as follows. Section 2 presents the task allocation and routing problem and simulation environment, section 3 describes the distributed architecture and software implementation, section 4 provides a description of the experiments conducted to provide the results of section 5.

2 Problem Formulation

2.1 Simulation Environment

The task allocation environment discussed in this paper is similar to semi-automated or fully-automated material handling systems, where the number of cargo transportation vehicles transport cargo from one place to another on guided and pre-defined paths within the environment. The map of the environment is modelled with nodes and links. Autonomous vehicles traverse links when moving between nodes. Each task has its own start and destination node in the environment. A task is defined as travel from a start node to destination node traversing interconnected links.

The task allocation problem consists of a fixed number of tasks and vehicles at the start of the simulation. The main objective of the task allocation process is to find the best task sequence and respective vehicles to complete the task sequence according to

the optimisation criteria. The objective of the task allocation process is to optimise or minimise the completion time of all the tasks at each rescheduling instance. The completion time of all tasks is called the *makespan*. This is achieved by searching the best task sequence for the available tasks and their respective vehicles, which are capable to complete them earliest. A mathematical description is further discussed in section 2.2.

At the start of the simulation nodes are generated randomly and selected vehicles are parked at randomly selected nodes. Furthermore, the start node for a task may not be the same as the start node for a vehicle. As a result a vehicle maybe required to travel from its initial parked node to the tasks start node, as illustrated in Fig. 1. Here the task start nodes (B, D, G), destination nodes (C, E, H) and initial parked nodes (A, F) are represented by triangles, stars and circles respectively. Each vehicle explores for a task to undertake, however selection criteria of our simulation is based on the incoming time (t_{AB} or t_{CD}) of the vehicles to the tasks start nodes, where incoming time is a function of a vehicles velocity at each path segment, traffic congestion and total distance of the selected path. All vehicles simultaneously work on their assign tasks and once a vehicle completes its assigned task it determines next most suitable task and moves to the appropriate tasks' start node. The task allocation process continues until all tasks are allocated.

Fig. 1. Simulation environment and initial setup of tasks and AGV's

2.2 Mathematical Formulation

An AGV travels from its start location to destination on a guided path consisting of nodes and links. Each path can be divided into a number of smaller path segments, called *links*. To avoid collisions AGVs travel at varying speeds set by SiPaMoP. Two different maximum speeds are set for empty and loaded AGV's.

Task completion time consists of two components, namely the travel time from the current location to the start node of the task called *transient time* and the travel time from the tasks start node to the destination node, called *task time*.

Assuming vehicle V_i is allocated to task T_j and the path segments to reach the task T_j start node is PR_{ij}. Next, the path selected the task start node to destination node is PP_{ij}. Both, PR_{ij} and PP_{ij} contain k_R and k_P path segments respectively. Therefore, the total completion time of the task T_j by vehicle V_i can be calculated using Eq(1). The start time of the first allocated task is assumed to be the same for all the vehicles.

$$TCT_j = \sum_{k=1}^{k_R} \left(\frac{PR_{ijk}}{V_E}\right) + t_L + \sum_{k=1}^{k_P} \left(\frac{PP_{ijk}}{V_L}\right) + t_U \quad (1)$$

Where, TCT_j is task completion time of T_j, V_E is maximum empty speed, V_L is maximum loaded speed, t_L is loading time, t_U is unloading time. Furthermore, the available task list is given by the vector $T = [T_1, T_2, T_{3,\ldots,} T_J]$ and available vehicles $V = [V_1, V_2, V_3, \ldots, V_R]$ where J is number of available tasks and R is number of available vehicles.

Now, if a vehicle Vi, completes N_i number of tasks, total traveling time of the respective vehicle is given by Eq(2):

$$TTT_i = \sum_{ni=1}^{N_i} \left[\sum_{k=1}^{k_R} \left(\frac{PR_{ijk}}{V_E}\right) + t_L + \sum_{k=1}^{k_P} \left(\frac{PP_{ijk}}{V_L}\right) + t_U \right] \quad (2)$$

Where, TTT_i is total traveling time of vehicle V_i, N_i is the number of tasks allocated to vehicle V_i.

The *makespan* of the schedule is given by Eq(3):

$$\text{Makespan} = \underset{v=1}{\overset{R}{MAX}} [TTT_v] \quad (3)$$

Since one AGV can perform only one task at a time, the start time of task $j+1$ by V_i should always be greater than the start time of task j done by the same AGV as given by Eq(4).

$$ts_{ij} < ts_{i(j+1)} \quad (4)$$

Where, t_s is start time of task T, ts_{hj} is the start time of task T_h of j^{th} vehicle.

2.3 Simultaneous Task Allocation and Motion Coordination Algorithm

Task allocation is performed using a single round auctioning process and motion coordination is done based on SiPaMoP approach [16]. Initially, the task sequence is generated randomly by the task generator. The first task is broadcast to all autonomous vehicles allowing them to place a future bid for the task. After each vehicle has returned their bid, a winner is determined and is allocated the first task. The second task is then broadcast, followed by bids from each vehicle and a winner selected. This auction process of broadcast task, followed by bidding and the selection of a winner continues until all tasks have been allocated. For each vehicle the calculation of bids is based on the travelling time to complete the current broadcast task and any previously allocated (as the winner) tasks.

Once the travel time has been calculated it is used to post bids ($B_{1,j}$ to $B_{i,j}$). The travelling time of collision free paths is calculated using the SiPaMoP algorithm. A winner is then determined, based on the lowest travel time to finish the respective task (completion time). When the completion time is calculated for each autonomous vehicle, its previous task commitments are also considered, which helps reduce the trapping of tasks to one vehicle. For example, if a previous task is allocated to a particular vehicle, then there will be less tendency for the same vehicle to win the next task. In addition, load balancing of the vehicles can be achieved partially. After

all tasks of the current task sequence is allocated, total completion time (*makespan*) is calculated. This process continues for a fixed number of cycles with a different task sequence generated randomly in each cycle. Eventually the best task sequence is selected, which provides the minimum *makespan*. The flow diagram of the simultaneous task allocation and motion coordination algorithm is illustrated in Fig.2. Here, the parallel computation of the SiPaMoP algorithm is shown occurring towards the middle of the process. It is during this stage that the motion and path is calculated for each vehicle independently using different computing nodes.

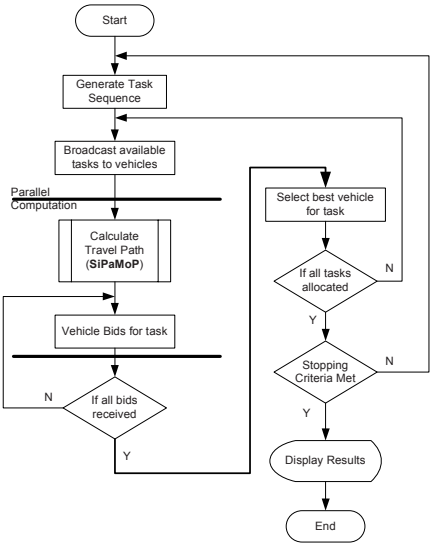


Fig. 2. Flow chart of STAMC algorithm

The travel path and resulting travel time to complete the path is calculated using the SiPaMoP algorithm. This portion of the STAMC algorithm is distributed and computed in parallel for each autonomous vehicle. If there exists, n autonomous vehicles requiring computation of a travel path, which takes t_{path} time to compute, the serial STAMC algorithm requires nt_{path} time to compute all paths for all vehicles. Whereas, the parallel STAMC algorithm requires only t_{path} time to compute the travel path for all autonomous vehicles.

A Master-Worker topology is used to distribute the STAMC algorithm. The Master node takes on the role of *Auctioneer*, by issuing tasks, receiving bids for tasks and determining the winning bid from each worker node. Worker nodes perform parallel computation of the path and motion planning using the SiPaMoP algorithm. The distributed computing environment and master-worker topology introduce a necessary communication time (t_{comm}), but $t_{comm} << t_{path}$.

Partitioning of the STAMC algorithm onto a parallel computing architecture is illustrated in Fig. 4 and the serial architecture is illustrated in Fig. 3.

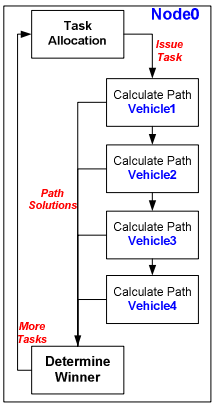


Fig. 3. The data path of *serial* computation for the task allocation and path planning algorithm for autonomous vehicles

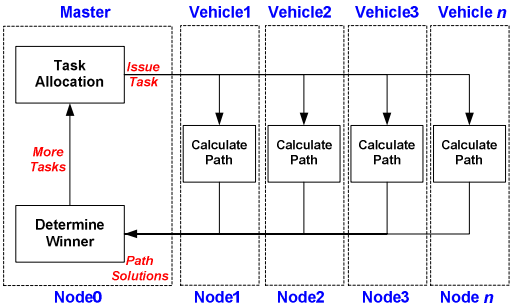


Fig. 4. The data path of parallel computation for the task allocation and path planning algorithm for autonomous vehicles

Here, the STAMC algorithm is implemented on a compute cluster, which is discussed further in section 4.2. The combined arrangement of hardware and software of the compute cluster provides a platform for coarse-grained parallelisation of the complete SiPaMoP algorithm, as illustrated in Fig. 4.

The cluster computing architecture and master-slave topology mutually provide for two possible mappings between the number of computing nodes and number of vehicles requiring motion and path planning. The first mapping is 1:1 and is used exclusively in this paper. Here, a single node computes the motion and path for a single vehicle. The second mapping, 1:*n* allows a single node to compute the motion and path for multiple vehicles. For example, if there exists, three computing nodes and nine vehicles, only three vehicles can be computed in parallel at one instance. As a result, three groups would be computed *sequentially*. In this case a single node would perform the motion and path computation for three vehicles, a 1:3 mapping. A third mapping of *n*:1, allows the motion and path computation to be further distributed across *spare* nodes. For example, if there exists, nine computing nodes and three vehicles it would be ideal to further distribute the path and motion computation for

each vehicle across the spare six nodes, thus allocating all computational resources of the compute cluster to the calculations. However, the STAMC algorithm encapsulates the complete SiPaMoP algorithm into a coarse-grained implementation, preventing any further decomposition into finer-grained portions for execution on separate processors.

The main objective for the simultaneous task allocation and collsion free path planning process is to minimse the *makespan* of the available tasks with the available autonomous vehicles. In order to satisfy the objective of task allocation, we must know the travel times of each tasks for respectable task-vehicle combination in advance and select the best task-vehicle combination. The SiPaMoP method is used to determine the travel time for each task and eventually we can find the best vehicle for the respective task. Since path planning and traveling times are determined by SiPaMoP method, paths are guaranteed to be collision free.

3 Integration of MPITB in the MATLAB Environment

The STAMC algorithm is implemented in the interpretive MATLAB environment, which has no native support for distributed computing. In order to arrange the STAMC algorithm into a Master-Worker topology on a distributed computing architecture a Message-Passing Interface (MPI) was required.

The integration of the MPI [17] and the interpretive MATLAB environment allows researchers to achieve coarse-grained and out-of-loop parallelisation of scientific and engineering applications developed in MATLAB. Developed at the University of Granada in Spain, MPITB for MATLAB allows researchers to include MPI function calls in a MATLAB application, in a way similar to the bindings offered for C, C++ and Fortran. Decomposition and coding of a serial problem into a parallel problem is still the responsibility of the researcher, as there is no *automatic* parallelisation method in MPITB. This method of explicit parallelisation coupled with user knowledge of the application provides a good chance for sub-linear or linear computational speedup. Furthermore, the onus is placed upon the researcher to develop *safe* distributed code free of livelocks and deadlocks which occur due to the loss of message synchronisation between distributed computing nodes.

Baldomero[18] provides a summary of several other parallel libraries that achieve coarse-grain parallelisation for MATLAB applications. The toolboxes differ in the number of commands implemented from the MPI specification and the level of integration with existing MATLAB data types.

The level of computational performance provided to a parallel MATLAB application is dependant upon the underlying communication method implemented in the toolbox. Toolboxes using the file system to exchange messages between computers tend to be slow due to the explicit read/write latency of rotating hard disks. Conversely, toolboxes using a message passing daemon to provide communication between computers provide much better performance due to the small latencies and large bandwidth capabilities of local area networks (LANs). In the later case, messages (data) are transferred between the primary memory of parallel computers, without buffering them using the file system prior to transmission over the LAN.

The MPI functions are written in C code and dynamically compiled into MATLAB MEX-files. The MEX-files encapsulate the functionality of the MPI routines, allowing them to be directly called within the MATLAB environment, thus making the parallelisation of the application possible. With both MATLAB and a message-passing library installed, such as LAM-MPI, the precompiled MEX-files can perform both MATLAB API calls and message-passing calls from within the MATLAB environment. The MPITB makes MPI calls to the LAM-MPI daemon and MATLAB API. This method enables message-passing between MATLAB processes executing in distributed computing nodes[18].

Transmission of data between the master-worker MATLAB processes and execution of the TAPP algorithm can occur after booting and initialisation of the LAM-MPI library from the master process using, `LAM_Init(nworkers,rpi,hosts)`. Where *nworkers* is the number of cluster nodes designated as worker nodes, *rpi* is the LAM MPI SSI setting which is set to either *tcp* or *lamd* in our experiments, *hosts* is the list of host names on the Linux cluster. Once the underlying MPI library has been initialised, MATLAB instances must be spawned on worker nodes. This is achieved using the `MPI_Comm_spawn(…)` command on the master process. Finally, establishing an MPI communication domain, called a *communicator*, defines a set of processes that can be contacted. This is done using the `MPI_Comm_remote_size(processrank), MPI_Intercomm_merge(processrank,0) and global NEWORLD` commands on the master process. Here, *processrank* is an integer greater than zero assigned to each worker node in the MPI communicator; the master node (processrank=0). The global variable *NEWORLD* is the name of the MPI communicator.

Transmission of messages between MATLAB processes can now be accomplished, permitting the arrangement of cluster nodes into any useful topology. The master-worker topology used in the experiments, employs the fundamental point-to-point communication mechanism between master and worker nodes, with one side performing a blocking send, `MPI_Send(buf,processrank,TAG,NEWORLD)` and the other a blocking receive, `MPI_Recv(buf,processrank,TAG,NEWORLD)`. When messages are sent *processrank* is the MPI rank value of the receiving process and when messages are received *processrank* is the value of the sending process. The parameter *buf* represents the MATLAB data to be sent or received, *TAG* is an integer associated with the message providing selectivity at the receiving node, and *NEWORLD* is the MPITB communicator. Any valid MATLAB data type can be transmitted directly without being prepacked into a temporary buffer, unless the message contains different data types. If the data to be sent is larger than a single variable, such as a matrix, then its size must be determined and sent prior to sending the matrix. The approach taken in this paper, is to calculate the size of the matrix in the sender using the MATLAB *size()* command, then send the size value to the receiver prior to sending the actual matrix. The distribution and parallel computation of the STAMC algorithm requires the transmission of a data between master and worker nodes. The size of the PATH_REGISTER matrix is dynamic between each task allocation cycle of the TAPP algorithm.

4 Experiment Description

The experiments involve execution of the STAMC algorithm on a single computing node and in parallel on the distributed computing cluster. The serial computation uses a single node of the Linux cluster, whereas the parallel computation uses multiple cluster nodes.

In this study, the performance is measured by recording the wall-clock time, so all components of the execution time, including communications, are included. The wall-clock time is a fair measure of performance that is frequently used.

4.1 Simulation Parameters

The set of algorithm and simulation parameters remained constant to encourage a meaningful comparison between the parallel/distributed approach and the serial/centralised method: *NumMaster*=1, *NumVehicles*={4,6,8}, *NumTasks*={24,48, 72,96,120,144,168,192,216,240}, *MapNodes*={192,216,240}, *WeightUpdate*=Dynamic, *VehicleSpeed*=100 cm/sec, *SchedulingTime*=1 batch, *TurnSpeed*=1 m/s, *SafetyTime*=0 sec, *TurningRate*=1.0. The STAMC algorithm is executed for 25 cycles with the average computation time taken as the as final result.

4.2 Cluster Computing Environment

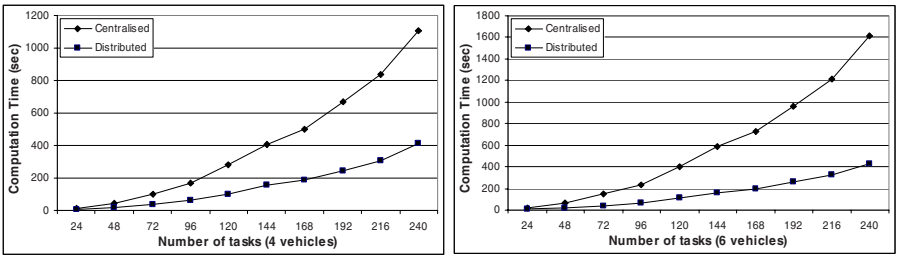
All experiments were performed on a Linux computing cluster with specifications provided in Table 1. The serial and parallel versions of the task allocation and path planning algorithm were coded in MATLAB. Communications between cluster nodes employed MPI 2.0 implemented using the MPITB.

Table 1. Cluster Computing Environment

Computing Environment Component	Description
Number of Nodes Utilised	1,4,8
Processor Type and core Speed	Pentium 4 @ 3.0Ghz (Prescott)
Front-side Bus Bandwidth	800MHz
DRAM capacity and bandwidth	2GB DDR @ 400MHz
Network Type and Bandwidth	1000Mbps Ethernet
Network Switching Type	Gigabit Switching Fabric
Network Protocol	TCP/IP V4
OS Kernel Type and Version	Linux (2.4.21-20.EL)
MPI Type and Version	LAM 7.1.1 / MPI 2
MATLAB	7.0.4.352 (R14) SP 2
MPITB	mpitb-FC3-R14SP1-LAM711.tgz

5 Results and Discussion

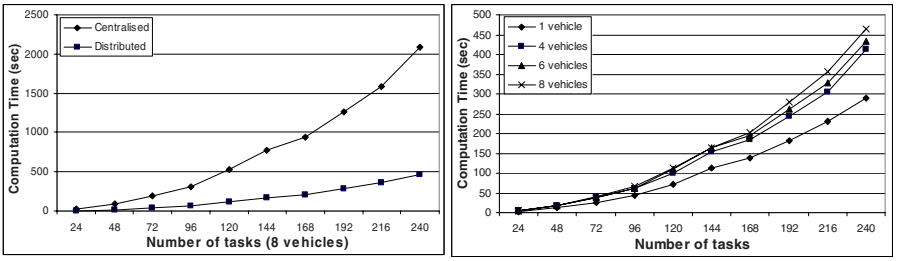
The average computation times for the parallel/distributed STAMC algorithm and the serial/centralised STAMC algorithm are illustrated in the Fig. 5 and Fig. 6.



(a) (b)

Fig. 5. (a) Computation time for parallel/distributed and serial /centralised STAMC with 4 AGVS; **(b)** Computation time for parallel/distributed and serial /centralised STAMC algorithm with 6 AGVS

From the three simulations using 4, 6, and 8 vehicles, there is a clear performance increase (less computation time) with the parallel/distributed STAMC algorithm compared to the serial/centralised STAMC algorithm. As the numbers of tasks are increased from 24 to 240, the difference in performance becomes even more significant between the two versions of the STAMC algorithm. In general, when number of tasks of the simulation study increases, the computation time increases exponentially. But the rate of change of the gradient in serial/centralised algorithm is much larger than the rate of change of the gradient in the parallel/distributed algorithm, because the STAMC algorithm is *evenly* distributed among cluster processors (1:1 mapping) and computed in *parallel* in the later case.



(a) (b)

Fig. 6. (a) Computation time for the parallel/distributed and serial/centralised STAMC algorithm using 8 AGV's; **(b)** Computation time for the parallel/distributed STAMC algorithm using 4/6/8 AGV's

Fig. 6(b) illustrates the results of the parallel and distributed STAMC algorithm using 4, 6 and 8 vehicles. Results for the single vehicle are also given to provide a baseline for comparison against the multi-vehicle simulations. For the multi-vehicle simulations, the computation time is similar from 24 to 240 tasks, with a maximum variation of approximately 14.3% between 8 and 4 vehicles at 216 tasks. This suggests a useful scalability property of the parallel STAMC algorithm arranged in a Master-Worker topology for an increasing number of vehicles.

The STAMC algorithm attempts to find an optimal (minimum) schedule for the allocation of *all* tasks to available vehicles, whilst guaranteeing collision free paths. This is a typical NP-hard ("Non-deterministic Polynomial time") problem, requiring time which is exponential in log *n*, the number of tasks to be scheduled. As a consequence the results are exponential in the number of tasks to be scheduled and not the number of vehicles. Because of the coarse-grained parallelisation of the SiPaMoP algorithm, variations between 4, 6, and 8 vehicles is small, even with 240 tasks as illustrated in Fig. 6(b).

6 Conclusion

The use of MPITB for MATLAB provided effective integration and encapsulation of MPI into the interpretive environment of MATLAB. This enabled existing MATLAB code to be distributed and computed in parallel using clustered computing power. Scientific and engineering applications continue to maintain the interactive, debugging and graphics capabilities offered by the MATLAB environment, and can now reduce the computation time by taking advantage of clustered computing.

Due to the high granularity of the STAMC algorithm, the distributed and parallel version achieved *near-linear* computational speedup over the serial STAMC algorithm. This result was achieved using 4, 6 and 8 cluster nodes for a number of tasks ranging from 24 to 240. The experimental results also suggest good scalability of the parallel STAMC algorithm, which becomes more important as the number of vehicles and number of tasks increases.

With the aim of increasing overall system reliability our future research work involves retirement of the master node to remove the single point of failure from the system. Coupled with a fully-connected topology, redundancy is increased by allowing any cluster computing node to adopt the master role of task allocation.

Acknowledgement

We wish to thank Dr Matthew Gaston for establishing, maintaining and supporting the UTS Engineering Linux Computing Cluster Environment.

References

[1] E. K. Bish , F. Y. Chen , Y. T. Leong, B. L. Nelson, J. W. C. Ng, and D. Simchi-Levi, "Dispatching vehicles in a mega container terminal *OR Spectrum,* vol. 27, Number 4 pp. 491 - 506 2005.

[2] J. Bose, T. Reiners, D. Steenken, S. Voß, and "Vehicle dispatching at seaport container terminals using evolutionary algorithms," *In: Sprague R H (ed) Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, DTM-IT, pp 1-10. IEEE, Piscataway*, 2000.

[3] M. Grunow , H. Günther, and M. Lehmann, "Dispatching multi-load AGVs in highly automated seaport container terminals, *OR Spectrum,* vol. 26, Number 2, 211-235 2004.

[4] J. K. Lim, K. H. Kim, K. Yoshimoto, J. H. Lee, and "A dispatching method for automated guided vehicles by using a bidding concept," *OR Spectrum*, vol. 25, 25-44, 2003.
[5] K. H. Kim and K. Y. Kim, "Routing straddle carriers for the loading operation of containers using a beam search algorithm, *Computers & Industrial Engineering,* vol. 36, Issue 1, pp. 109-136, 1999.
[6] P. H. Koo, W. S. Lee, and D. W. Jang, "Fleet sizing and vehicle routing for container transportation in a static environment, *OR Spectrum,* vol. 26, Number 2, 193 - 209 2004.
[7] R. L. Moorthy and W. Hock-Guan, "Deadlock prediction and avoidance in an AGV system," *Master of science, Sri Ramakrishna Engineering College, National University of Singapore*, 2000.
[8] F. Xu, H. V. Brussel, M. Nuttin, and R. Moreas, "Concepts for dynamic obstacle avoidance and their extended application in underground navigation, *Robotics and Autonomous Systems,* vol. 42, Issue 1, pp. 1-15, 2002.
[9] A. I. Corréa, A. Langevin, and L.-M. Rousseau, "Scheduling and routing of automated guided vehicles: A hybrid approach, *Computers & Operations Research,* 2005.
[10] R. Eisenberg, R. Stahlbock, S. Voß, and D. Steenken, "Sequencing and scheduling of movements in an automated container yard using double rail-mounted gantry cranes, *Working paper, University of Hamburg,* 2003.
[11] A. Wallace and "Application of AI to AGV control - agent control of AGVs," *International Journal of Production Research*, vol. 39(4), pp. 709-726, 2001.
[12] A. I. Correa, A. Langevin, and L. M. Rousseau, "Scheduling and routing of automated guided vehicles: A hybrid approach, *Computers & Operations Research*, vol. to be published, 2005.
[13] T. Le-Anh and M. B. M. De Koster, "On-line dispatching rules for vehicle-based internal transport systems, *International Journal of Production Research,* vol. 43, Number 8 / April 15, 2005 pp. 1711 - 1728 2005.
[14] G. Desaulniers, A. Langevin, D. Riopel, and B. Villeneuve, "Dispatching and Conflict-Free Routing of Automated Guided Vehicles: An Exact Approach, *International Journal of Flexible Manufacturing Systems* vol. 15, Number 4, pp. 309 - 331, 2003.
[15] A. K. Kulatunga, D. K. Liu, G. Dissanayake, and S. B. Siyambalapitiya, "Ant Colony Optimization based Simultaneous Task Allocation and Path Planning of Autonomous Vehicles, *IEEE International conference on Cybanatics and Information Systems 2006, Bangkok Thailand*, pp. 823-828, 2006.
[16] D. K. Liu, X. Wu, A. K. Kulatunga, and G. Dissanayake, "Motion Coordination of Multiple Autonomous Vehicles in Dynamic and Strictly Constrained Environments, Proceeding of the *IEEE International conference on Cybernetics and Information Systems* (CIS), 7-9 June, 2006, Bangkok Thailand, pp204-209.
[17] MPI.Forum, "MPI-2: Extensions to the Message-Passing Interface," in *Message Passing Interface Specification*, vol. 2005, November 15, 2003 ed: NSF and DARPA, 2005, pp. MPI-2 Specification Document.
[18] J. F. Baldomero, "Message Passing under MATLAB," presented at Advanced Simulation Technologies Conference, Seattle Washington, 2001.