

UNIVERSITY OF WESTMINSTER



## **WestminsterResearch**

<http://www.wmin.ac.uk/westminsterresearch>

### **An open interface for parallelization of traffic simulation.**

**Damian Igbe**  
**Nasser Kalantery**  
**Stephen Ijaha**  
**Stephen Winter**

Cavendish School of Computer Science, University of Westminster

Copyright © [2003] IEEE. Reprinted from 7th IEEE International Symposium on Distributed Simulation and Real-Time Applications, 23-25 Oct 2003, Delft, Netherlands.

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the University of Westminster's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to [pubs-permissions@ieee.org](mailto:pubs-permissions@ieee.org). By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

---

The WestminsterResearch online digital archive at the University of Westminster aims to make the research output of the University available to a wider audience. Copyright and Moral Rights remain with the authors and/or copyright owners. Users are permitted to download and/or print one copy for non-commercial private study or research. Further distribution and any use of material from within this archive for profit-making enterprises or for commercial gain is strictly forbidden.

---

Whilst further distribution of specific materials from within this archive is forbidden, you may freely distribute the URL of WestminsterResearch. (<http://www.wmin.ac.uk/westminsterresearch>).

In case of abuse or copyright appearing without permission e-mail [watts@wmin.ac.uk](mailto:watts@wmin.ac.uk).

# An Open Interface for Parallelization of Traffic Simulation

Damian Igbe, Nasser Kalantery, Stephen Ijaha, Stephen Winter  
Centre for Parallel Computing (CPC)  
University of Westminster  
115 New Cavendish Street  
London, W1W 6UW, United Kingdom.  
{d.o.igbe | kalantrn | ijahas | wintersc}@wmin.ac.uk

## Abstract

*In this paper, we present the implementation of a parallel road traffic simulation using the concept of Lane Cut Points (LCPs) in the Spider programming environment. LCPs are storage buffers inserted into lane data structures at the road network partition edges. Vehicles enter a partition at the edges from an LCP and exit a partition edge into an LCP at the end of every simulation step. Spider, a parallel programming environment, which runs on PVM, coordinates the execution of the parallel traffic simulation.*

## 1. Introduction

Traffic simulators can be classified as either macro or micro simulators based on the level of detail of the state variables considered in the simulation. For macro simulations, the details of the simulation entities such as vehicles, junctions, traffic lights, intersections, driver behaviours, lane changing are not taken into consideration while all these are considered in micro simulation. For example, individual vehicle movement representation is a characteristic of typical micro-simulators, whereas in macro simulators only traffic flows are requested. While the macro method does not give a good report about the behaviour of each vehicle at any point in time, it offers simplicity of simulation and its use of system resources are minimal. Micro simulation, on the other hand, can give operative details on the traffic, which is useful to the traffic engineer but introduces more complexity in terms of programming and requires greater computation power and storage space.

Large-scale micro simulation models therefore can be time consuming when executed on single processors. A way of speeding up the execution is to use a cluster of workstations. A cluster presents itself as a good choice because of its low cost, high availability and scalability. To run on a cluster the simulator must be “parallelized”. To parallelize the simulator, the road network data must be partitioned and sent to each node. Necessary exchange of data during execution requires coordinated communication.

In this paper, we present our experience of the parallelization of “Madcity”, a microscopic urban traffic simulation. Related work is introduced in section 2. The experimental model of the parallel traffic simulation is discussed in section 3. In section 4 the parallelization strategy is discussed. Conclusions and further work plans are discussed in section 5.

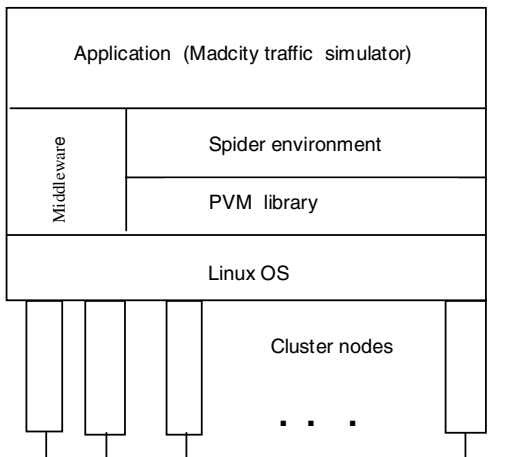
## 2. Related work

Much research is presently aimed at parallelizing traffic simulators. Some of the related work can be found in Transims [1] [14], Paramics [2] [12], OSSA [10] and Hipertrans [4] which are European Union projects and the parallel traffic simulation based on Dynemo model presented in [6].

All the above-mentioned simulators are based on microscopic modelling and their parallelization strategy, as in our method, is by using domain decomposition concept, where the road network is partitioned into subnetworks and these subnetworks are executed on different computers. This method gives better scalability than that of functional decomposition, where a limited number of functional components of the simulator are executed on different computers [5].

### 3. Experimental Madcity parallel traffic simulation system

The architecture of the parallel Madcity simulator is shown in Fig. 1. The major components are discussed below.



**Figure 1. The architecture of the parallel simulator**

#### 3.1. The Parsifal cluster

The hardware platform is a cluster of 32 nodes running Linux Operating system. A cluster is defined as a collection of interconnected computers that can be used as a single unified resource for high performance computing [3].

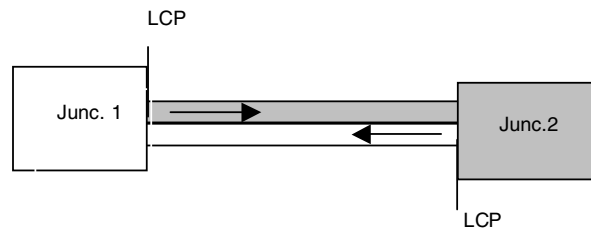
Cluster technology promises to provide the benefits of supercomputers at a significantly lower cost. However, extra communication overheads present on a cluster means that achieving the desired performance may require extensive parallel program optimisation expertise that may not be available to an application programmer.

#### 3.2. The Madcity microscopic simulator

Madcity, a discrete time based microscopic simulator, is organized around a compound data structure that represents the road network. The road network is modelled as a collection of interconnected *junctions*. *Junctions* are interconnected through *roads* where each *road* may have multiple traffic *lanes*. Traffic is represented as a set of *vehicle objects* that use the road network. The road network also contains representation of traffic control equipment (e.g. traffic lights). At each step of the simulation, each vehicle uses a set of simple localized rules to compute its new state and new

position. In this, a vehicle must take into account its surrounding conditions. For example, the proximity to a slower vehicle ahead will influence the speed of the vehicle, etc. The ability to read the surrounding conditions is made available through the network structure. The overall complex pattern of the urban traffic emerges from the simple local actions of the individual vehicles.

From a user point of view Madcity comprises two major components; the Graphical User Interface (GUI) and the Simulator kernel (SIM). In addition to providing a GUI for road network modelling, Madcity GUI also incorporates a Network Partitioning Tool (NPT). NPT allows the user to partition the network by assigning a partition ID to each junction. A partition ID can then be saved along with the rest of the road model to be used by the SIM. This is illustrated in Fig. 2. As the figure indicates, NPT uses junction Partition IDs at two ends of interconnecting road to decide whether the road is to be divided or not. If the two junctions have non-identical Partition IDs then the road must be partitioned as “cut across”. An issue here is where exactly the road lanes must be cut. NPT is designed to allow the user to explicitly position the *Lane Cut Points* by graphical means. However, it also provides a default option where a road is partitioned according to the following simple rule; a road interconnecting two junctions is partitioned such that each exit lane belongs to its origin sub-net. Lane Cut Points are discussed further in section 4.1.



**Figure 2. A simple 2-junction network illustrating the use of LCP**

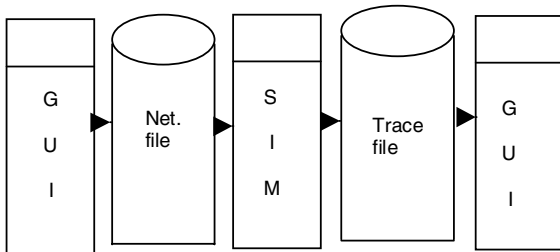
In Madcity, partitioning is junction based, each junction belonging to a given sub-net. At the end of this process each junction will have a given partition identifier. Roads do not have explicit partition identification. The junction data structure maintains a list of its entire exit links.

Junctions with identical partition IDs constitute a sub-network. Each partition is represented with colour coding around the junction. This ensures that partitioning is assisted with an adequate visual interface and remains a user-friendly process that is well integrated into the road network model.

To simulate a road network, a user has to go through the following procedures:

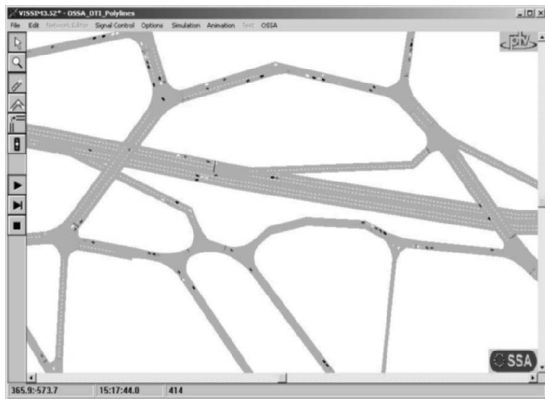
- Use the GUI to draw a road network and partition the network
- Save the road network in a “network” file
- Load the network file into the SIM module
- Run the simulation, this produces a trace file
- Feed the trace file into the GUI module, which carries out the visualisation.

These procedures are illustrated in Fig. 3 below.



**Figure 3. The main simulation stages**

Fig. 4 shows a screen dump from the simulation of the city of Hyde in UK. The visualization in this case is done using the OSSA visualizer [10], rather than the Madcity GUI.



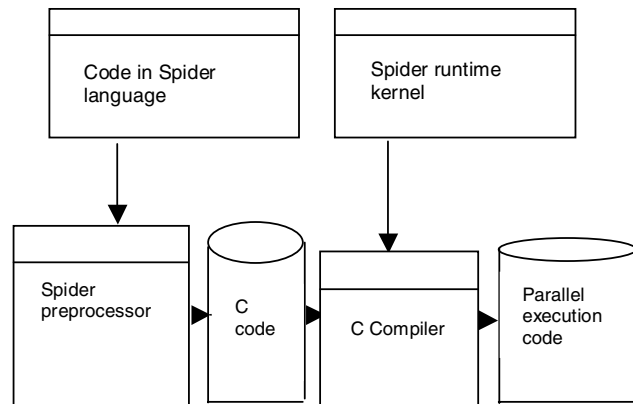
**Figure 4. Screen dump of the Hyde network during a simulation**

### 3.3. The Spider programming environment

Spider, which is built upon PVM [13], is a cluster-based parallel programming environment, an implementation of the Virtual von Neumann machine [8] that has been developed at the University of Westminster. Spider adopts a discrete event based approach to program parallelization [9] and supports a

synchronous mode of parallel computing which is suitable for road traffic simulation.

The Spider system consists of a pre-compiler unit and a run-time kernel. The pre-compiler ensures that application programs can be expressed in a convenient high level language. This language is a super-set of C, where global network control constructs have been added to the usual repertoire of uni-processor control statements. These include shared variable declarations and distributed loop constructions. Using the construct ‘MultiLoop’ the user can specify a set of parallel loops. These loops can exchange messages asynchronously (through Mget/Mput statements) or use shared variables to achieve an automatically synchronised communication (through the assignment operator ‘:=’). Spider makes use of a logical time-stamping mechanism whereby access to shared data is treated as a discrete event, i.e. a point in a two dimensional space-time co-ordinate. This ensures that transparent synchronisation is achieved without using extra communication, a key technique by which efficient and scalable parallel performance is obtained. The pre-compiler accepts application programs and produces a suitably structured code that includes explicit calls to the run-time kernel.



**Figure 5. The illustration of Spider program stages**

The run-time kernel provides the underlying facilities through which process creation, distribution, communication and synchronisation are realised. These stages are illustrated in Fig. 5. Conceptually, the Spider kernel resides on a cluster of workstations and provides convenient global control abstractions and parallel programming facilities to the user. These include virtual global memory, virtual global clock, allocation, distribution, creation and termination of parallel processes. The kernel supports multiple models of parallel computation. It supports both *implicit* and *explicit* parallelism. It can be used to automatically

parallelise a sequential loop, or create *Distributed Shared Memory* and/or *direct message passing* applications. It supports both synchronous and asynchronous programming paradigms as illustrated in Fig. 6.

	IMPLICIT	
E X P L I C I T	Synchronous	Asynchronous
	Distributed Shared memory	Direct Message Passing

**Figure 6. The Spider modes of Operation**

#### 4. Parallelization strategy

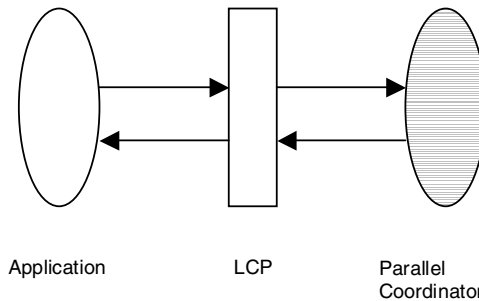
The method of parallelization and the structure of the parallel program are discussed in the following sections.

##### 4.1. Lane Cut Points (LCPs)

An important step in parallel processing is decomposing the problem or data into different tasks to be distributed to the clustered processors, for the simultaneous executions. A consideration here is to keep partition sizes as even as possible. These partitions are then distributed amongst multiple nodes of the cluster for simultaneous execution.

Once a network is partitioned into concurrent sub-simulations, two major runtimes issues must be dealt with; communications and synchronisation. Assuming that these two issues are resolved, a coordinated execution of the whole simulation will be achieved. However, organizing and implementing synchronisation and communication requires expertise that an application programmer may not necessarily possess. It would be useful to have a mechanism whereby the parallel coordination concerns could be separated from application development issues, such that, two different sets of entities could meet across a common interface and yet be able to work independently, each in their own familiar area. The concept of LCPs was developed to serve such a purpose. LCP is a data structure, which encapsulates vehicle data at the partition edges and

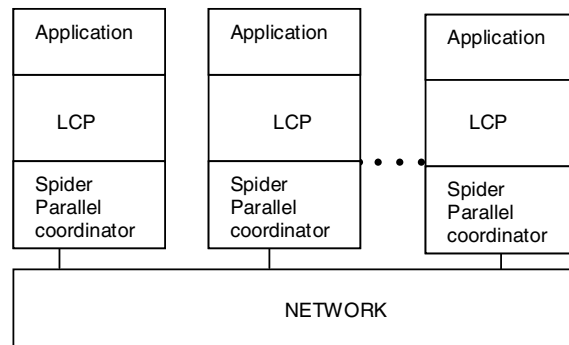
discrete-time synchronisation is achieved using LCP. The LCP concept is depicted in Fig. 7.



**Figure 7. The illustration of LCP concept**

In decomposing a road network into multiple sub-networks, traffic continuity is potentially disrupted. During runtime, when a vehicle arrives at the boundary point in a lane, where one subnetwork ends and another one begins, the vehicle must be transferred seamlessly to the computing node where the vehicle can continue its journey. This vehicle transfer must be coordinated in time and space such that the integrity of the simulation remains intact. The LCP interface in conjunction with parallel coordination must meet these requirements.

##### 4.2. The Madcity parallel co-ordinator



**Figure 8. Illustration of Spider as a Madcity parallel coordinator**

The coordinator is illustrated in Fig. 8. The madcity coordinator is a program written in Spider language. The coordinator defines and maps a deriving loop for each partition of the simulation and specifies local processing as well as exchange of non-local data that should take place during the distributed execution. Whenever a spider program is executed, the following steps take place:

- The language construct “Begin” starts up copies of the program on all available nodes of the underlying parallel virtual machine.
- An initial handshake ensures that each of these instances wait until all of them have been created and are ready to execute.
- After the handshake each instance of the spider executes the local code until a multiloop statement is encountered. A key parameter in the multiloop statement is its index range (a multiloop can be viewed as an array of driver loops). Using the index-range each node of the Spider virtual machine uses a simple round robin algorithm to decide whether one or more statements of the multiloop maps on this code. If so, a corresponding task frames for each locally mapped local interface is created.

A spider program may have one or more multiloop specified in it. The above procedure continues until instances of all the declared multiloops are mapped on all the nodes of Spider Virtual Machine (SVM). This continues until a “commit” statement is encountered. Commit forces the SVM code into execution mode where created task instances are run. SVM maintains a Ready-To-Run queue (RTR) and a suspended queue for task frame management. Whenever a task gets blocked on an input operation, then it is moved into the suspended queue. Tasks that are in the RTR are executed in round-robin fashion, where a virtual time slice allows each task to execute no more than  $n$  iterations before it is moved to the back of the RTR queue and the task at the head of the RTR is executed.

### 4.3. The structure of the parallel program

The programming model used is SPMD (Single Program Multiple Data). This means that the same program image is available on all of the nodes but each of the nodes has a different data set. The data sets in this case are the different portions of the partitioned road network. SPMD programs are easier to write and maps perfectly into the Spider programming model.

The local program has three different phases:

The initialisation phase. This is more of a sequential code since it does all the necessary initialisations that apply to all the nodes. These include issues such as:

- Reading the network graph and partition identifiers from network file
- Sending the network data across to the nodes.
- Using the network data to find LCP positions and creating them as required

The local execution phase. In this phase, each node uses its system ID to identify itself, and its portion of the network and then starts the simulation of the road traffic.

At each simulation step:

- Each partition is simulated concurrently
- Vehicles leaving a partition are passed to the LCP interface.

The communication phase. In this phase, coordinator picks up the vehicles in the LCPs and sends them across to their respective partitions where they are inserted into the appropriate lane for the simulation to proceed normally. Synchronous operation is achieved by use of Spider communication constructs at the end of every simulation step. Vehicles leaving a partition need to be received in their destination partition before the next simulation step.

### 4.4. Advantages of LCP-based coordination

There are four main benefits.

1. These LCP standard interfaces provide a facility whereby decomposition of a simulation is developed and debugged on a single processor environment and hence complexities of debugging in a parallel environment are significantly reduced. It is necessary here to differentiate between a serial simulator and the simulator making use of the interface objects such as LCP but running on one node. Use of the LCP library helps to develop an  $n=1$  system that has all the features of parallel simulator. Thus whenever vehicles, which are meant for the neighbour partitions are leaving a partition, these vehicles enter the LCPs and then removed from the LCPs at the end of every simulation step. In this way, the behaviour of the parallelized simulator can be analysed in the  $n=1$  version and the transition from the  $n=1$  to the  $n=k$  ( $k \geq 2$ ) parallel development effort is therefore minimised since most of the possible errors can be debugged in the  $n=1$  version.

2. LCPs give the advantage of separating the application layers from the communications/coordination layers that is required for the parallel version of the simulator. LCPs provide an open interface such that a given simulation application could be coupled with alternative communication layer and vice versa, such that different simulations could use the same LCP interface for parallelization purpose. To accomplish this objective, the simulator developers need to develop their applications to conform to the interface definitions published in the LCP documentation.

Also, use of LCPs can reduce the communication overhead associated with parallel programming. Instead of sending individual vehicles across the partitions, vehicles meant for a particular partition can be buffered in the LCP and all the vehicles transferred at once to their respective partitions. Communication overhead, which has a negative effect on the performance of a parallel system, can therefore be minimised by use of LCPs.

3. At certain intervals, all the nodes involved in running a program will need to exchange certain entities, vehicles in this case. Though the use of LCPs means that every vehicle has to be monitored to know when they need to get into their neighbour partitions, it rather fits into the method of microscopic simulation where the behaviour of individual vehicle is important. LCPs fit into this simulation model because synchronisation takes place at the end of every simulation step, whereby vehicles are always checked to determine their current position and also to make such decisions as whether the vehicles should go into or out of an LCP.

4. Finally, LCP technology is not difficult to implement and so individual developers can easily adapt it to their programs.

The fewer the number of LCPs however, the better the performance of the simulator. For maximum efficiency, the point at which the network partitions are made should be to minimise the number of lanes crossing the partition boundaries. This reduces the number of LCPs in the network and minimises the overhead in processing the vehicles in the LCPs at the end of every simulation step.

## 5. Conclusions and future work

This is an ongoing research effort to develop a high performance microscopic traffic simulator. The Spider programming environment has been described. The concept of Lane Cut Points (LCPs) that are inserted at the end of lanes between partitions was also introduced and discussed. LCPs provide an open interface whereby transformation of a sequential simulator to a parallel one is facilitated.

The overall aim of this project is to achieve a high performance traffic simulator and to test the efficiency of using Spider for parallel road traffic simulations. In due course, the performance of this system is under investigation.

At the moment, partitioning of road networks is done manually via the NPT interface. This will be augmented with an automatic static partitioning tool, where a user draws the road network without assigning any partition IDs to the network but the user enters the number of desired partitions in a configuration file. The static load balancer will scan through the network making use of the configuration file and assigns the partition IDs.

In addition, since traffic simulation presents an irregular load of traffic because of vehicles moving from one partition to another, a way of optimising traffic simulations is to perform load balancing dynamically. It is therefore intended to build a dynamic load balancer integrated into the system in the near future.

## 6. Acknowledgements

This research is being carried out as part of the OSSA (Open Framework for Simulation of Transport Strategies and Assessment) project DGTREN GRD1-10917, within the Competitive and Sustainable program of the European Communities. The parallelized simulator will be tested along other modules being developed in the OSSA consortium by other technical partners. We would like to thank all the other technical partners involved in this project for their collaboration and discussions.

## 7. References

- [1] C. Barrette, and D.J. Roberts, "The TRANSIMS micro simulation status", Technical report, TSA-DO/SA, Los Alamos National Lab, New Mexico, USA, 1994.
- [2] D. McArthur, *The PARAMICS Model: Present and Future Directions*. Technical report, SIAS Ltd., Edinburgh, 1994.
- [3] Greg Pfisters, *In search of Clusters*, Prentice Hall, 1998.
- [4] Ijaha S.E, S.C. Winter and N.Kalantery, "HIPERTRANS - High Performance Transport Modelling and Simulation", Proc, 6<sup>th</sup> Intl Europar Conf., Munich, Germany, 28 August - 1<sup>st</sup> September, 2000.
- [5] Kai Nagel and Marcus Ricket, "Parallel implementation of the TRANSIMS micro-simulation" <http://www.inf.ethz.ch/personal/nagel/papers/parallel/parallel.pdf> [Online: cited March 23, 2003].
- [6] Matthias Schmidt, "Decomposition of a traffic flow model for a parallel simulation", Proc. AI, Simulation and planning in High Autonomy Systems (AIS2000), Tucson, USA, March 6-8, 2000.
- [7] Message Passing Interface (MPI) Standards, <http://www-unix.mcs.anl.gov/mpi/index.html> [Online: cited August 8, 2003].
- [8] N. Kalantery, S.C. Winter and D.R. Wilson, "From BSP to a virtual Von Neumann Machine", IEE journal of Computing & Control Eng., vol 6, no 3, June 1995, pp 131-136.
- [9] N. Kalantery, "A distributed event processing method for general-purpose computation", J. Systems Architecture, no 44, 1998, pp 547-558.
- [10] Open Framework for Simulation of Transport Strategies and Assessment (OSSA), A European 5th Framework Collaborative Research Project in the Competitive and Sustainable Growth Programme, DG-TREN Project GRD1-10917 "OSSA", <http://www.ossa-ig.com/>. [Online: cited August 8, 2003].
- [11] Parsifal Cluster page: <http://parsifal.cpc.wmin.ac.uk/> [Online. Cited August 8, 2003].
- [12] Paramics: <http://www.paramics-online.com> [Online: cited August 8, 2003].
- [13] Parallel Virtual Machine (PVM) Library, <http://netlib2.cs.utk.edu/pvm3> [Online: cited August 8, 2003].
- [14] TRANSIMS, TRansportation Analysis and Simulation System, <http://transims.tsasa.lanl.gov> [Online: cited August 8, 2003].