Appears in the *Proceedings of Supercomputing'94* (Washington DC, USA)
© IEEE Computer Society Press, pp. 291-300, November 1994.

PARAMICS — Moving Vehicles on the Connection Machine

Gordon Cameron [*]
Edinburgh Parallel Computing Centre (EPCC), The University of Edinburgh

Brian J.N. Wylie [†]
CSCS, Lugano, Switzerland

David McArthur
SIAS Ltd, Edinburgh

Abstract

PARAMICS is a PARA*llel* MIC*roscopic Traffic* S*imulator which is, to our knowledge, the most powerful of its type in the world. The simulator can model around 200,000 vehicles on around 7,000 roads (taken from real road traffic network data) at faster than 'real-time' rates, making use of 16K processor TMC Connection Machine CM-200 for the simulation aspect. The project aims to make available to road network planners a new range of tools, and demonstrates that use of high performance computing in real applications is possible and worthwhile, while yielding important and interesting research results.*

1 Introduction

This paper describes the PARAMICS microscopic traffic simulation package, designed and written at the Edinburgh Parallel Computing Centre (EPCC) in conjunction with SIAS Ltd, and part funded by the Science and Engineering Research Council (SERC) and the Department of Transport.

PARAMICS is implemented making use of a Thinking Machines Corporation(TMC) CM-200, configured with 16K processors and a DataVault fast storage device. The project aim is to implement a system whereby individual vehicles can be simulated accurately on all the major trunk roads in the Scottish road network, and updated at real time rates. In this use, 'real time' means that the simulator should be able to simulate the movement of vehicles in a certain time, and that this time should be equal to (or less than) the *real* time it would take the vehicles to progress that distance. The 'time-step' for the simulator update is one second of real 'driver time', so the simulator itself should also take one second at most for each iteration. The number of vehicles simulated is of the order of 200,000 on around 7,000 road links, and to our knowledge, this is the biggest simulation attempted at this complexity anywhere in Europe. At time of writing, only one other group that we are aware of in the world is looking at such a problem size([8]), although interest is growing in this area.

[*]Contact the author at `gordonc@epcc.ed.ac.uk`
[†]Work done whilst at EPCC

In particular, our *own* interest is in studying the congestion problems in the Scottish trunk road network. However, it should be noted that any network for which information is available can be modelled (even if the network is in countries where people drive on 'the other' side of the road!).

2 Background

2.1 PARAMICS : What is it, and why ?

While traffic simulation programs have been in existence for many years, they have all had to use simplified models of traffic flow in order to produce results within practical timescales. A typical assumption is to represent traffic flow on a particular road as a single quantity, analogous to electricity flowing though one link in a circuit. Such models are generally classed as being *macro*scopic simulations. Unfortunately, such models do not properly represent real traffic behaviour in congested situations, and do not reproduce the inherently fluctuating nature of real world situations.

PARAMICS[1, 2] is a *micro*scopic simulator, as we are interested in accurately modelling congestion formation and dispersion, as well as maintaining an accurate picture of what is actually happening. It has been argued that congestion formation is a phenomenon associated with the chaotic, non–linear nature of road traffic and as such is best modelled at the microscopic level [3], modelling individual vehicles on each of the road links. Each time step sees the vehicles shunted along the roads, and moved from road to road if relevant, as in real-life. (Of course, microscopic simulators can also gather statistics on flows through roads, so these figures can be compared with results from the macroscopic simulations — see §9.)

2.2 Previous Work

The PARAMICS project is based on a sequential system named MICSIM (Microscopic Simulator)[5], which was developed as part of project IMAURO[6]. MICSIM could simulate an urban network of around 50 junctions and 200 vehicles at real-time speed. PARAMICS is required to simulate around 200,000 vehicles at near real-time speeds, needing a thousand-fold increase in processing speed on the original simulator.

The timeliness of the PARAMICS project is evident from the number of other projects attempting traffic simulation using parallel computers. Examples include the TRANSIM project [7, 8], issues of traffic control being studied using arrays of transputers [9] and the recent work published by K. Nagel et al[10, 11, 12]. Many other groups are just staring research in this area, and given the current direction in road traffic policy in Scotland, the project is seen as being particularly timely[4].

3 Parallel Environment

The machine used for the project was a 16K processor CM-200 based at EPCC. An explanation of the SIMD paradigm is outwith the scope of this article, but in such a model of programming, performance is gained by utilising a very high number of simple processors connected in a tightly coupled network, executing the same code strictly in lock-step (Single Instruction), but each having their own areas of data on which to operate (Multiple Data).

The language chosen for implementation of the simulator was C* [13], an extension of C which enables the programmer to make use of the CM-200 by writing essentially sequential-looking programs with instances of array-type constructs, the elements of which can be distributed on different processors, and operated on in parallel. These arrays are usually of a size which is a multiple of the *physical processor size* of the machine (where the arrays are larger, this is handled transparently by the machine). Each single notional element is known as a *virtual processor*, and operations occur in parallel on these virtual processors.

To aid in loading and saving data to/from the CM-200, the DataVault fast storage device is used. This is a farm of disks which 'appears' as a normal storage medium to the SIMD program, the difference being that it is used to store distributed data. That is, parallel variables present on the CM-200 can be stored in a special format on this very fast I/O device, so that they can be reread *directly* back on to the machine without having to go first through the front end. The CM-200's only connection to the 'outside world' is through the front-end workstation(s).

4 Data Issues

4.1 Data Available

4.1.1 Network Data

Network data available consists of a graph description (*nodes* and *links*) for the entire major Scottish trunk road

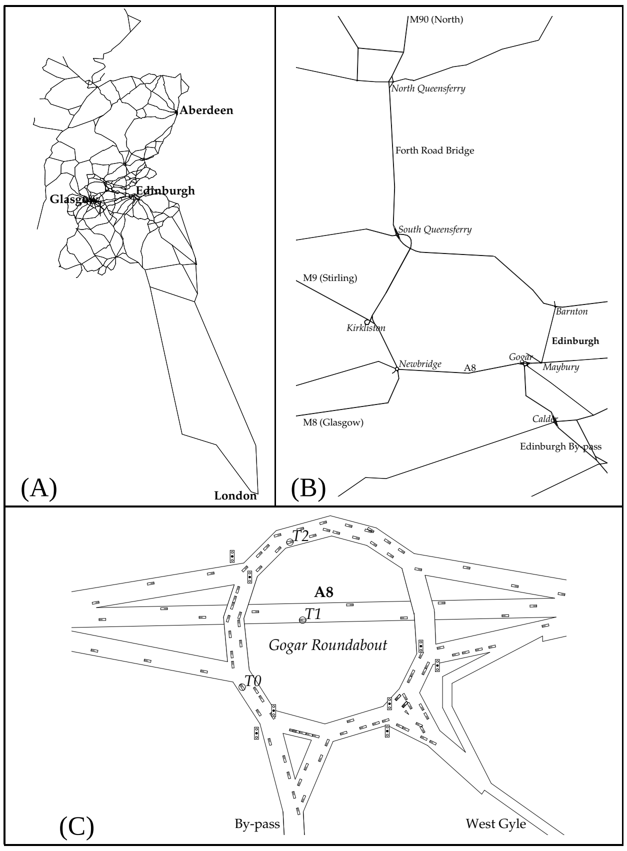

Figure 1: PARAMICS network and traffic visualiser displays

network. Each node-to-node connection is a uni-directional link, and there are two links (in opposing directions) for each road. Each road has an associated type, which gives information on attributes like the number of lanes, maximum speed allowed, length and curvature of the road etc. The graph itself represents the connectivity of the road network (Figure 1, diagram **A**).

The network has been refined during the PARAMICS project to include more detail on specific junctions and roundabouts [1] (this can be seen in Figure 1, diagram **C**).

4.1.2 Other Available Data

Other data to which we have access to, or have created, includes:

- **Routing Information** from any given link in the network graph to any of several destination *zones* (points where vehicles leave the trunk road network). The routes are calculated statically, and represent the 'shortest paths' in terms of length of links, speed possible, etc;

[1] Roundabouts, for the benefit of readers in countries that do not have them, are simply a road layout allowing easy exit from a given link to several others without needing complex junctions or flyovers - see Figure 1

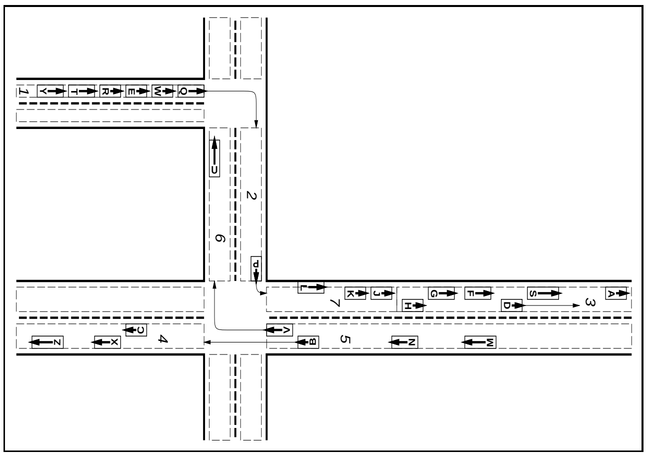

Figure 2: Traffic network extract

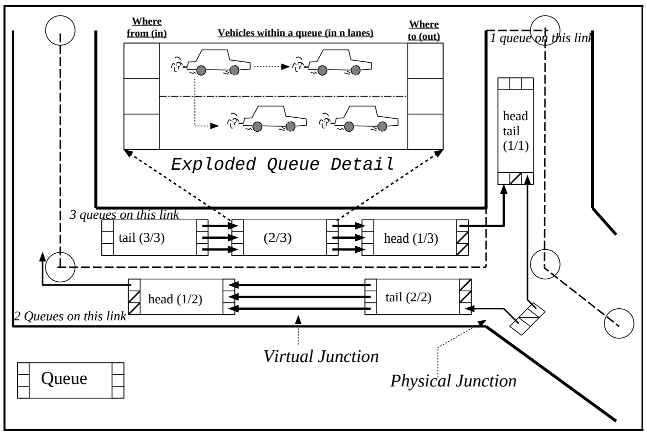

Figure 3: Queue segments in the network

- **Snapshot information** in the form of flow figures on links at typical times;
- **Vehicle Release Rates** at points in the network;
- **Vehicle Characteristics** detailing classes of typical vehicle;
- **Traffic Light Phasing** representing the light sequencing at certain locations.

All the information that we have access to, and wish to model, has to be first translated into a microscopic form suitable for use by a microscopic simulator (by, for example mapping flow rates to actual numbers of vehicles - see §5.1), and then mapped into a parallel form that is suitable for use on the CM(see §5.2).

4.2 Parallelising the Data

The approach we settled on for building a parallel data framework for the simulation process was to associate with each of the uni–directional links (see §4.1.1) a *queue*. The *queue* is the parallel item of data on which we hope to operate on the CM-200(§3). Figure 2 shows an extract of the network traffic, with the queues outlined using dashed lines and numbered. Each of these segments exists on a separate virtual processor, and, among other things, holds details on the vehicles present on that segment. The bold lines show the 'real' physical underlying network, which is not needed by the simulator. The parallel array is a 1-D array consisting of a large number of these queues.

4.2.1 Queues

Each queue can be thought of as first and foremost a container class that contains a wealth of information, including:

- an array of vehicle slots, into which can be placed vehicles currently on that queue, regardless of lane,
- connections detailing queues ahead, in all possible directions (i.e. at junctions), and to queues behind, from which vehicles will join,
- priorities at the junctions,
- routing information in the form of turn tables. These are lookup tables which, given a vehicles's destination zone (see §4.1.2) can determine which direction the vehicle must turn at the junction to help move towards its destination

The most important thing to remember is that each of these queues is controlled by a *different* virtual processor on the CM-200 , and so operations on these queues can be performed in parallel. Another point to remember is that each queue can control vehicles on all of the lanes that comprise the link.

Forcing each link to having one queue is somewhat limiting, as each and every queue in the simulation can then only hold at most the same amount of vehicles as every other queue, since the array of vehicle slots are the same size on each processor. Hence, the speed of the simulation will be limited by the number of vehicles held in each queue, and each queue must have enough space to be able to hold all the vehicles on its corresponding link. This makes for extremely bad load-balancing, where many of the queues will have slots for far more vehicles than they have vehicles, hence will be idle a lot of the time.

It is obvious that many links (or roads) will have a far greater capacity for vehicles than others, so we introduce the concept of additional floating queues which join at *virtual junctions*. Each queue in the simulation has the same number of vehicle slots, but links are now allowed to have more than one queue (Figure 3). The problem of where to assign (extra) queues to links is covered in §5.2 and §6.5, but in the meantime, there are three important points worth making about queues :

1. The front queue is known as the *head queue*, and connections out from this queue cross a *physical* junction. Similarly, the last queue in a link is known as the *tail queue*, and connections *into* this queue also cross a

physical junction. Note that each link needs at least a head and a tail, but that the two can be one and the same queue.

2. Connections into all queues that are *not* tail queues, and out of all queues that are *not* head queues form *virtual junctions*. These only occur when there is more than one queue on a link, and the world 'virtual' is used to show that no real junction exists between the non-head queue and the queue ahead (Figure 3).

3. Although queues are associated with links in the traffic network, in the cases where there is more than queue on a link, the participating queues 'float' in space. That is to say, each queue is *not* limited to looking after a spatial portion of the link. *Queues are simply containers for vehicles, and it is the* vehicles *in the vehicle slots of each queue that have a spatial position.* The only spatial queue enforcement made is that vehicles in a given queue are spatially ahead of vehicles in the queue behind on the same link.

In addition to the above, a wealth of other parallel data is kept, stored within the queue structure. One of the most important is the note kept at each non-head queue of the distances ahead to tail vehicles in the queue in front. Information is also held on: whether or not the queue has been allocated and is being used (see §6.5); the number of vehicles currently controlled by the queue (see below); statistics on how many vehicles of each type the queue has controlled, and more.

4.2.2 Data Held on Vehicles

Every queue has nV vehicle slots into which vehicles can be placed (Figure 5). Where the slots are not empty, a vehicle exists, and each vehicle holds with it a wide variety of information about itself. A slot is simply a temporary container which can hold details on a vehicle, or be empty where no vehicle is present. More specifically, each vehicle in the simulation has information on:

- **Vehicle type**, which defines things such as physical size, maximum speed, acceleration decceleration, absolute target speed, vehicle class (e.g. bus, heavy goods vehicle, domestic car, etc);
- **Destination** in the form of a zone;
- **Dynamically changing values** such as current speed, distance along link, current lane, etc.

As the simulation progresses, this information moves around from slot-to-slot and queue-to-queue, representing the vehicle itself moving through the traffic network.

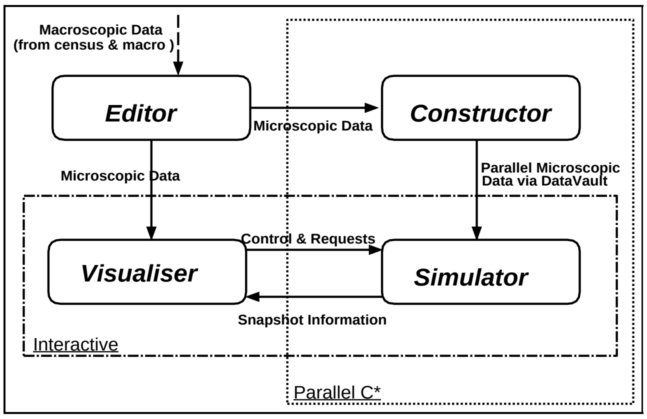

Figure 4: Diagram Showing Dataflow in the PARAMICS Project

5 Software Components of PARAMICS

The PARAMICS project consists of four main software components : Editor, Constructor, Simulator and Visualiser. Figure 4 shows the data flow relationships between these pieces of code, along with details on how they interact and share information. The following sections describe the functions of two of the components, with the simulator and visualiser being addressed in detail in §6 and §7 respectively.

5.1 Editor

The editor is used to transform data from the essentially macroscopic information supplied by SIAS's other simulators and the network data supplied by The Scottish Office (see §4.1.1), into a microscopic form suitable for use by the PARAMICS simulator. This data, once in microscopic form, can be mapped onto the parallel data framework by the constructor. The editor need typically be run once only to set-up files for a given data network.

5.2 Constructor

The constructor is used to parse the sequential (microscopic) data generated by the editor (§5.1), map this to parallel variables, and store the results. This preparation of data stage is time consuming (of the order of hours) given that there is such a large amount of data to process. Hence, the constructor is distinct from the simulator as it need typically setup data only once.

The queue structure, as described in §4.2.1 is built and initialised by the constructor. It is the constructor that works out how many queues each of the links in the network data will get, based on average flows on the link (from census and macroscopic flow data), and on the physical length of the link. The constructor also decides on the number which will define how many vehicles each queue can hold (i.e.

the number of vehicle slots in the queue structure). This number especially, together with the total number of queues that the constructor allocates will in part dictate the load balancing and efficiency of the simulation.

Note that, since congestion is a moving phenomenon, the initial distribution of queues to links should be later updated, as the number of vehicles on links (and hence queues) fluctuates. In addition, the heuristics utilised by the constructor to allocate queues may result in the long term in artificial congestion (where all the queues in a link are full, even though physical space may exist on that link). Since it is not feasible to allocate as many queues as to completely saturate the network, of a size big enough to cope (this would give horrendous load balancing), dynamic queue reallocation and reshaping through time is necessary to allow real traffic simulation to progress. This has been addressed, and is detailed in §6.5.

The constructor places vehicles initially on the network according to snapshot data for the time of day of interest, and once it has set up all the parallel data, this is written to the DataVault, where it can be later quickly read by the simulator, allowing the simulation process to commence rapidly.

6 Simulator

6.1 Overview

The simulator component of PARAMICS comprises the main parallel computational element. Its job is essentially to move vehicles around the road network as realistically as possible, taking account of other vehicles, crossing priorities, traffic lights, safe distances and so on, whilst potentially communicating with the visualiser, and updating other dynamically changing features.

6.2 Main Simulator Functions

The following describes, in a simplified manner, the main loop being executed in the simulator — remember that this is being done on every virtual processor concurrently :

- Read in initial configuration of queues and vehicles from DataVault, as set up by the constructor
- Loop (for required number of timesteps)
  - Loop (over vehicles in each queue in parallel)
    - Move vehicles along queue (§6.3)
  - Rerank vehicles in queue
  - Exchange vehicles between queues (§6.4)
  - Perform dynamic queue reallocation (§6.5)
  - (Listen/Send information to/from visualiser - §6.6)
  - Update traffic lights (§6.7)
- Exit

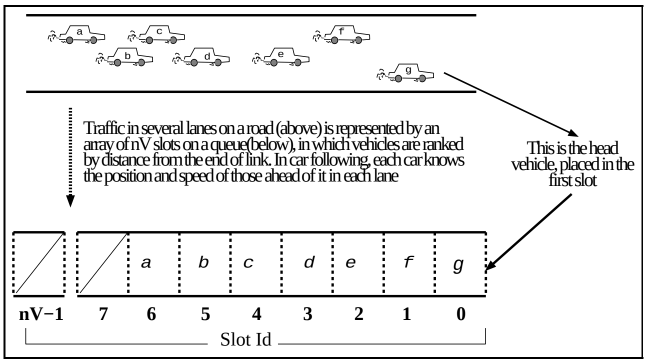
a
c
f
b
d
e
g
Traffic in several lanes on a road (above) is represented by an array of nV slots on a queue(below), in which vehicles are ranked by distance from the end of link. In car following, each car knows the position and speed of those ahead of it in each lane
This is the head vehicle, placed in the first slot
a
b
c
d
e
f
g
nV−1
7
6
5
4
3
2
1
0
Slot Id

Figure 5: Mapping of vehicles to slots

6.3 Car Following

Car following is the most computationally intensive part of the simulation process. In this stage, all vehicles in the simulation are moved along the links with which the containing queue is associated, and have their attributes updated in accordance with any movement. Note that no communications between queues needs to be done during this phase, which is described below (where nV is the number of slots in each vehicle queue, and nL is the maximum number of lanes in a road) :

```
FOR loop=0 TO nV-1
  IF there is a vehicle in slot 'loop' THEN
    FOR lane=0 TO nL-1
      # ~ PART A ~ set the ahead structure
      IF 'lane' < num. lanes on this queue
        eval conditions ahead of vehicle in lane
      # ~ PART B ~ driver behaviour
      change the vehicles lane, (if relevant)
      adjust the speed of the vehicle
      update the  ahead structure for this vehicle
# ~ PART C ~  update vehicle position
FOR loop=0 TO nv-1
    IF there is a vehicle in slot 'loop' THEN
        move vehicle forward, based on speed, etc.
```

Note that the loops are performed over all the slots in a queue, so that on processors where not all the slots contain vehicles, there will be some idle time while the loop completes.

During the previous timestep of the simulation, vehicles within slots were ranked by distance along the link. Hence, slot 0 holds the vehicle nearest spatially to the end of the current link, so the loops start with this vehicle and work backwards. (see Figure 5)

6.3.1 Part A — Setting the lookahead structure

A running structure is kept of the statistics of vehicles in each of the possible lanes (there are nL), and this is known as the *ahead* structure. Using this structure, each vehicle knows how much space there is ahead to the vehicle in front (if any) in each of the lanes, and the characteristics of such vehicles. Specifically, the ahead structure stores details for each lane such as: current tailpoint of the vehicle ahead (if any), type of the vehicle ahead, speed of vehicle ahead, etc.

This ahead structure is initialised to be empty for head queues, for the first iteration. As each vehicle is looped over, that vehicle adds its own details to what was there already, thus keeping an updated view.

For non-head queues, the ahead structure is set to the completed ahead structure for the *previous timestep* of the queue in front, across the virtual junction. This means that vehicles in non-head queues know the tail positions and speeds of the trailing vehicles in the queue ahead, across the virtual junction, so can still set their behaviours accordingly. This involves communication between queue segments, and is done during the vehicle exchange phase.

6.3.2 Part B — Driver Behaviour and Attribute Update

Each vehicle must evaluate what its target speed will be for the current timestep, what acceleration or decelaration it will have to apply to get to that target, and which lane it should be in. It can make such decisions based, essentially, on four factors :

- Details of distances to the vehicles ahead in each lane, and their speeds and vehicle types, etc;
- Distance to the end of the road (and hence a physical junction);
- Priority at the end of the road;
- Details on the vehicle's own *current* characteristics.

In the case of free flowing traffic, each vehicle will attempt to travel at a speed closest to its outright target speed (defined by type) but allowing a safe distance to vehicles ahead. Vehicles will, by applying acceptable accelerations and decelerations, reach a figure which determines the speed for the next time step, where the process will repeat. Hence, the speed of vehicles will change from step to step, with each vehicle attempting to reach its outright target speed.

In the case of slower moving traffic, or traffic approaching a physical junction (marked by the approach of the end of the link), speed is potentially tempered somewhat more. In the case of a major priority junction, the vehicle need not be slowed very much, but in the case of a barred junction (for example the red phase of traffic lights) more deceleration should be applied to slow the vehicle. Minor priority junctions have a similar but less severe affect on the vehicle speed. (Note that vehicles need not be slowed additionally when merely approaching *virtual* junctions, which float in space).

As far as lane changing is concerned, a vehicle will attempt to change into an outside lane if that lane will allow them to travel closer to their type's target speed (i.e. faster), and this is classed as overtaking. In addition, depending on the turn that the vehicle wishes to make at the end of a junction, it will aim to move into the 'correct' lane for that turn.

6.3.3 Part C — Update vehicle position

After the first loop has completed, and all the vehicles are in their required lanes, and have had their target speeds set, the second loop moves all the vehicles along their link based on the speed they were assigned. An additional check is made to ensure that no 'accidents' have occurred(!), through vehicles inadvertently overlapping. In such cases, the speeds and positions are readjusted.

6.4 Vehicle Exchanges

When a vehicle reaches the head of a head queue, and the end of a link, it is time for the vehicle to move onto a different queue. This portion of the simulation involves communication between queues, which must exchange details on vehicles.

In each timestep, vehicle exchanges are performed, moving vehicles from original or *source* queues, to desired *target* queues, by :

- Identifying those vehicles that wish to move onto another queue (source queue perspective)
- Arbitrating amongst conflicting requests to join a queue (target queue perspective)
- Transferring vehicles that have been accepted (target queue perspective)
- Removing successfully exchanged vehicles from their original queue (source queue perspective)

Note that in the simulation the identity of the 'current' queue being examined and operated on changes from being the source queue when nominating vehicles, to the target queue for arbitration and transfer, back to the source queue when removing succesfully transferred vehicles, and that *at each timestep of the simulation, at most one vehicle can leave any given queue*.

6.4.1 Nominating vehicles for transfer

Any vehicle that has reached a head queue, and is in the first slot of that queue, and has reached the junction, is nominated for transfer. The identity of the vehicle is sent to the queue which the vehicle wishes to join. Each queue holds with it a turn table detailing turns (left, right, straight ahead) necessary to move vehicles onto their particular target destination zone (see §4.2.1). In addition, each queue *also* knows which other queues lead out from it in each of the three directions — left, right and straight ahead (Figure 3).

Vehicles waiting to move onto another queue can look up the turn table to see which way they need to turn to get to their destination (rather like looking at the road sign at the side of the road). Using this information, the vehicle can then determine *which* specific queue it wants to join, moving from the original or *source* queue, to the desired *target* queue.

In addition, *all* vehicles in the first slot of non-head queues are nominated for transfer to the queue ahead, as such vehicles are attempting to perform a virtual crossing only.

6.4.2 Arbitration

After all the nominations from queues have been performed, arbitration is done from the perspective of queues which have vehicles waiting to join them. A vehicle is accepted on the queue iff there are slots available in the queue, and there is physical space between the tail vehicles and the end of the link (or vehicles in the queue behind).

In addition, there may be more than one vehicle vying to join the queue, and in such a case the vehicle that has major priority across the junction is selected. In cases where this is not clear or there is more than one vying vehicle with major priority, one of those is chosen at random for transfer.

6.4.3 Vehicle Transfer

Target queues that are to accept vehicles ask for information on the vehicle from the source queue, and update their own statistics, as well as that of the vehicle they receive. Source queues are informed that the vehicle they wished to transfer has been accepted (or not, as the case may be).

6.4.4 Cleanup after vehicle exchange

After the vehicle transfers have been completed, source queues that nominated vehicles for transfer have been informed if the send was successful. If so, then details of the vehicle are removed from the queue.

6.5 Dynamic Queue Reallocation

Since congestion is a dynamically changing phenomenon, the distribution of queues to links, as done by the constructor (see §5.2), is likely to be inadequate as time goes on. This can lead to some undesirable behaviour such as artificial congestion occuring where the simulator does not have enough queues, and hence total vehicle slots, to hold all the vehicles that can legitimately be on a link. The most extreme case is where deadlock occurs in the network, as congestion tails back over the entire system.

To try and remedy this situation, we have a devised a scheme where a notional pool of 'free' queues is kept, to which unused queues in the network are returned, and from which queues can be taken to add to links that require more capacity for vehicles. This pool is represented as a list of queue identifiers (or *qids*) on the front-end, the *qids* referring to queues that are currently unallocated and hence available for reallocation, and the queue pool is initialised to contain all those queues that are unallocated to links by the constructor.

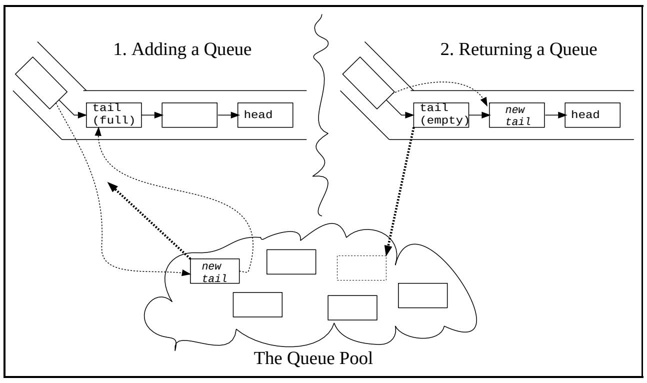

Figure 6: Dynamic Queue Reallocation using Queue Pool

Every 10 timesteps, the network of queues is evaluated to check where there are links that need extra queues, and those available free queues are added to the links. In addition, every 10 timesteps, but offset by 5 from the check for new allocation, unused queues are returned to the queue pool.

6.5.1 Adding New Queues

A simple test is performed to see where new queues are required. Where all the vehicle slots in a tail queue are completely full, those queues request a new queue to be added to the link, behind themselves. The front-end arbitrates between requests for new queues, and the available queues in the pool, and sends to each of the succesful requesting tail queues the *qid* of the queue that is to be added at the end of the link. This new queue will become the new tail.

The existing tail queue can then 'pretend' that it is the new tail queue, by setting all the pointers in and out of itself, and other attributes (whilst storing in temporaries the old values). It can then send itself to the new queue, whose *qid* it has received from the front-end, and then set itself back to the values it had previously. This means that only one general send needs to be done, and makes for efficient use of the machine.

Finally, the queues which pointed to the old tail are informed that they now point to the new added tail(Figure 6, diagram **1**).

6.5.2 Returning Queues

The test for whether or not a queue is returned to a pool depends on several factors. First, the queue, to be returned, must be a tail queue and must have all its vehicle slots empty. Second, there must be more than one queue associated with the link on which it exists (otherwise we could remove the one and only queue on a link, thus rendering that path unpassable in future). Thirdly, the queue ahead must also be less than half full of vehicles.

If all these criterion are met, the queue informs the front-end of its *qid*, reinitialises all its fields, and marks itself as being unallocated. The front-end then adds the *qid* to the pool, marking it available for reallocation elsewhere.

Finally, the queues which pointed to the tail that has just been removed are now set to point to the new tail, which is the queue ahead of the one removed. (Figure 6, diagram **2**).

6.5.3 Gains of Dynamic Reallocation

We observe that the performance penalty of adding code to the simulator to do dynamic reallocation is minimal, and that the gains, in terms of more accurate simulation, are significant.

6.6 Communications with Visualiser

When the simulator is executing in tandem with the visualiser, the two communicate via non-blocking message passing, with an intermediate buffer space between the two, allowing a certain degree of slackness.

The visualiser can request that the simulator stop and start, and also ask for vehicles (or flows) from certain areas of the network to be sent. In such a case, a parallel mask is constructed to select queues within the area that is of interest to the visualiser. Each timestep, vehicles which are within this mask are gathered onto the front-end, placed into messages, and despatched to the visualiser. The interaction between simulator and visualiser can become somewhat complex, but further discussion is outwith the scope of this paper.

6.7 Traffic Light Update

At each timestep, the phasing on groups of traffic lights is updated. These lights control exits from specific links to other links, and when the light at a particular junction changes phase/colour, the junction prioirities out from the head queue on the relevant link must be updated. For example, when a light goes red at junction, the outgoing priorities from the link to the other links which the link controls must be set to be barred, so that no vehicles can 'run' red lights !

7 Visualiser

Having a good means of showing what is happening is useful for several reasons, including: browsing of data (static and dynamic); watching the simulation process; debugging; offline examination of data; controlling the simulator, and so on.

The PARAMICS visualiser is a separate process, executing on a Silicon Graphics (SGI) workstation, which is designed to run either concurrently with the simulator, or in a standalone mode. The same code and program is executing in both cases, but when the simulator is also present, data can be extracted and the simulator controlled. In such a case, the simulator and visualiser communicate making use of EPCC's multi-platform message passing library, CHIMP [14]. When executing stand-alone, the visualiser can replay simulation snapshots at leisure, extracting the same types of data that were available when the data was produced.

Some of the features include: interactive display of network, vehicles, annotation, text, flows and other information; fast data navigation; control of the simulator; grabbing and visualising data from the simulator; recording/playback of vehicle snapshots from previous simulations; accurate roads, junctions and vehicle motion; tracking of vehicles; in-betweening of vehicle frames; image and PostScript saving of views; separate zoom and plan windows; support for animation, video capture, and so on.

Figure 1 shows some sample output. **A** shows a general overview of the network modelled, **B** shows the area of specific interest in the study, and **C** displays one of the key roundabouts with vehicles on the links. (The vehicles are displayed with lines showing the direction of travel, and intended turn at the end of the link). The last figure also shows traffic lights on the roundabout, and examples of some traced vehicles, marked **T0**, **T1** and **T2**.

8 Performance

8.1 General Performance

One of the goals of the project was to produce a system which could simulate traffic at real-time rates, on a 16K node CM-200, and this has been met and exceeded resulting in a simulation process which can simulate the required 200,000 vehicles on 7,000 roads at faster than real-time. Moreover, we can achieve this performance: executing the visualiser in tandem with the simulator, grabbing information as required; on *half* the Connection Machine i.e. 8K nodes; on a typically loaded Ethernet; without recourse to obscure machine-specific low-level coding.

8.2 Performance Improvement

Any comparison with existing serial simulators is bound to be at best subjective, and at worst, misleading. However, when compared to, for example, the original microscopic simulator MICSIM[5], which made use of a DECstation, we can observe a speedup factor of around 1,000.

In a sense, this is missing the point. PARAMICS was designed to solve a very large problem, with a required performance. Such performance would have been impossible using a sequential machine, and we believe that the simulation system we have designed is well suited to data parallel architectures, and makes efficient use of the Connection Machine. In addition, we had to expend relatively little effort to achieve such high performance, in terms of tweaking the code to use low-level routines, and we believe that this will make the code easily portable to other architectures which support the data-parallel programming paradigm.

8.3 Performance Details

To aid in program development, debugging and profiling, use was made of the TMC debugging tool PRISM. Profiling code, and inspecting results within PRISM enabled us to isolate portions of the code that were responsible for the majority of the time, and improve performance.

As well as profiled timings, we could make measurements based on wall clock timings — since we know how long each timestep *should* take, we can *observe* the machine performance by watching both the visualiser, and timestep output.

8.3.1 Data Initialisation

Data initialisation is performed whenever the simulator is started up, and accounts for a constant amount of time (around 20 seconds). In this time, the simulator loads in and verifies data from the DataVault, and sets up all structures that need initialising. In addition, the visualiser must read in data on initialisation, but when the tools operate in tandem, the startup times are almost identical — when the visualiser is ready for use, the simulator has loaded in all data from the DataVault, setup all data, and is ready to simulate.

It should be noted that, without the use of the Constructor (see §5.2), data reading and initialisation would take several hours. In addition to data loading, much of the initialisation process takes place on the front-end to the Connection machine, and *not* on the CM itself.

8.3.2 Simulator Performance

The simulator itself seems to work extremely efficiently. For example, using profiled code, on half the Connection Machine (8K processors) and running for 50 timesteps, the CM time is around 39 seconds *of which over 35 seconds are spent doing computation*.

Allowing the simulation to proceed for around 4000 timesteps (or just over an hour), and subtracting the initialisaton time (which accounts for a a miniscule proportion of the time on longer runs anyway), the general communications on the machine tend to take around 3-5 percent of CM time *only*. In this time, *all* vehicle exchanges are being done, *all* updates of lookahead structures are exchanged, and *all* CM communication in the dynamic queue segmentation is being done.

Once the simulation process commences, wall clock timing show that the simulation process procedes faster than real-time — managing around four timesteps every three seconds.

8.3.3 Visualiser Performance

The visualiser tends to run at interactive rates (as can be seen in the video), even when running in tandem with the simulator, and even when displaying typical numbers of vehicles and flows.

Non-blocking message passing is used between visualiser and simulator, and interaction on the visualiser always takes precedence over communications.

8.3.4 Bottlenecks

The main bottlenecks seem to occur on the Connection Machine front end. If this machine is heavily loaded, then performance degrades. In addition, there is only one ‘wire’ between the front-end and the CM, and if other users are using this path, or too much information is passed along this wire, then performance also degrades.

Data initialisation on the front-end accounts for a large proportion of the time on small runs of the simulator, but is a constant startup cost.

When running in tandem with the visualiser, the performance that can be observed on the visualiser is usually highly interactive, but depends somewhat on the loading on the Ethernet network used. In addition to simply simulating vehicles, the simulator has to strip off data to the front-end, and package this up and send it across the Ethernet. Note that this is not an issue when the simulator executes stand-alone.

9 Verification

We are currently in the process of verifying the behaviour of PARAMICS, by comparison with observed traffic patterns, and results obtained on flow from existing, used, macroscopic systems. Initial results have proved promising, and we are contintuing to improve the simulator with this verification in mind.

10 Conclusions

The PARAMICS project has produced a system that can perform extremely fast and detailed microscopic simulation on a much larger scale than previously possible, and yielded a suite of software tools that will be further developed in future projects.

We believe that data parallel programming can be a very effective way to solve such microscopic simulation problems, and that good use can be made of machines such as the CM-200 to such an end.

It is hoped that future work will lead *both* to tools that can aid road network planners, and research into the various issues surrounding parallelisation as well as into specific areas such as driver behaviour.

Work continues on the PARAMICS project on the CM-200, and areas being investigated include,

- **Extension of the driver behavioural model**, slotting in behavioural 'units' [15].
- **Integrating RTI information**. **R**oad **T**raffic **I**nformation technologies range from signs placed on overhead gantries informing drivers of road conditions ahead, to roadside beacons which transmit information on congestion, accidents, incidents and other traffic information to a receiver sitting inside the vehicle [4].
- **Environmental Modeling**. Much interest is being shown, [8], in modelling the impact of vehicles on the environment, and microscopic modelling enables the likes of emissions to be accurately modeled.
- **Incident/Accident Modeling**. More complex events such as accidents, flooding, roadworks, temporary closures and so forth could be handled.
- **Dynamic Re-routing**. Using a calculation strategy to allow vehicles to dynamically change the way in which they will traverse the road network to reach that destination is, of course, essential for the modelling of RTI systems, incidents and accidents, where the driver has information which may enable them to alter their path.
- **Feedback**. Some prototype road devices are available for obtaining actual real-time statistics on flows on roads, and these devices may be able to feed directly into the simulator.
- **Decreasing Time-Step**. We are experimenting with different smaller time steps – for example, half-second timesteps seem to give more realistic results.

In addition to the above, at time of writing we have just completed an initial MIMD version of the simulator as part of a project primarily concerned with porting the simulator to a Cray T3D. The redesign has resulted in a portable message passing simulator implemented using the MPI standard. It is hoped that using this version we can attempt to gain extremely high performance (thus allowing pre-emptive simulation in tandem with 'real' road systems), and be able to run the simulator on a much wider range of platforms.

11 Acknowledgements

We would like to acknowledege the help of the following people and institutions : *Gordon Duncan, Mark Smith, Matthew White* (EPCC); *Malcolm Brown and Harvey Richardson* (TMC); *Stephen Druitt* (SIAS), and *The Scottish Office* for the data.

References

[1] B. J. N. Wylie, G. Cameron, M. White, M. Smith, and D. McArthur, "PARAMICS: Parallel Microscopic Traffic Simulator." in proceedings of EuroCM User Group meeting, Paris, 1993.

[2] G. Cameron, M. Smith, M. White, B. J. N. Wylie, and D. McArthur, "The PARAMICS Parallel Microscopic Traffic Simulator," in *UK IT Forum Conference Digest*, 1994.

[3] P. Toint and P. Dehoux, "Some comments on Dynamic Traffic Modelling in the presence of Advanced Driver Information Systems," in *Proceedings of the DRIVE Conference (Brussels)*, pp. 964–980, Elsevier Science Publishers, February 1991.

[4] The Scottish Office, "Where Now? A National Driver Information and Control Strategy for Scotland." Consultation Document distributed to relevant and interested parties, Feb. 1993.

[5] D. McArthur, "Functional Specification of the MICSIM Sub-Model," Project report V1014, DRIVE Office, Brussels, Dec. 1989.

[6] "Final Report of the IMAURO Project," tech. rep., DRIVE Office, Brussels, Feb. 1992.

[7] S. Rasmussen. Personal Correspondence.

[8] D. J. Roberts and C. Barrett, "The TRANSIM Microsimulation Status." WWW online document, detailing progress at LANL, 1994.

[9] S. B. Richardson and J. B. G. Roberts, "Large Scale Simulation of Road Traffic on Multi-transputer Machines," in *Proceedings of the European Simulation Multiconference (Copenhagen, June 17–19, 1991)*, 1991.

[10] K. Nagel and A. Scheicher, "Microscopic Traffic Modelling on Parallel High-Performance Computers," *Parallel Computing*, vol. 20, pp. 125–146, Jan 1994.

[11] K. Nagel and M. Schreckenberg, "A Cellular Automaton Model for Freeway Traffic," *J. Physique I*, 1992.

[12] K. Nagel and A. Schleicher, "Microscopic Traffic Modeling on Parallel High Performance Computers," Report 93.132, Universität zu Köln, Mathematisches Institut und Zentrum für Paralleles Rechnen, Koln, Germany, May 1993.

[13] Thinking Machines Corporation, Cambridge, MA-02142, USA, *Programming in C**, Version 6.0.2 ed., June 1991. Incorporates the 'C* Programming Guide' and 'C* Users Guide'.

[14] J. G. Mills, L. J. Clarke, and A. S. Trew, "CHIMP Concepts," tech. rep., Edinburgh Parallel Computing Centre, 1991.

[15] D. McArthur, "A Rule Language for Describing Driver Behaviour in an IRTE," in *Proceedings of the DRIVE Conference (Brussels)*, pp. 1488–1498, Elsevier Science Publishers, Feb. 1991.