Special-Purpose Parallel Computer for Traffic Simulation

H. J. M. van Grol and A. F. Bakker

Traffic simulations are widely used for long- and medium-term forecasting of traffic. Now-a-days, with the growing problem of queues during rush hours, the demand arises for dynamic traffic management and, therefore, short-term forecasting. Apart from the need for new, adapted dynamic assignment models the second important part in this new development is the required computational power. Most commercially available computers are unable to produce an accurate forecasting for the next 5 to 30 min. within the desired time and budget. Analysis of existing assignment models and their most time consuming part—shortest path finding—has shown that the main structure of the models can be parallelized. The use of parallelism thus seems apparent. Several general purpose parallel computers, such as N-cubes, are commercially available. However, apart from being more expensive, they loose a large part of their expected performance by the amount of necessary interprocessor communications. Additionally, the programming of such computers has turned out to be more difficult than expected. A simple linear array of typically 16 processors, the so called Linear Processor Array (LPA) is proposed. This one-dimensional parallel computer with high-speed buffered interconnections between each pair of neighboring processor boards, parallel accessible by both a control board and a general-purpose host computer, forms a transparent concept for the programmer. The optimally configured boards together with the high speed intercommunication allow a cost/performance improvement factor of 100 compared with a minisupercomputer like the Convex C1. LPA should be a powerful tool for future developments of on-line traffic control, route guidance, ramp metering, among other things.

Many transportation planning and control activities involve traffic simulations for medium- and long-term forecasting. The existing models are efficient enough to do this within the desired time. With larger networks, the simulation time increases rapidly. Furthermore, with the growing importance of dynamic traffic management (because building new roads is considered not to be a desired solution anymore), the demand for short-term forecasting grows. Existing models do not meet these demands, neither in time nor in accuracy.

Apart from the development of new adapted dynamic assignment models, the time constraints require thought about the necessary computing power. With most commercially available computers it is not possible to produce an accurate forecasting for the next 5 to 30 min within the desired time and budget. The use of special-purpose computers will open ways to a desired solution.

A cost-effective answer to this problem is given in this paper. To start with, the existing assignment models and the development of the new dynamic models will be investigated. This will lead to a simplified representation of assignment models in which the algorithm is analyzed to find possible optimizations. It will be shown that the main structure of the models can be parallelized. After a short discussion about the use of general-purpose computers and an introduction to special-purpose computers, the use of special-purpose hardware for traffic routing problems will be discussed. This will lead to the proposed Linear Processor Array (LPA), which will be explained in detail. Finally, some preliminary results will be presented and the expected performance improvement will be given in conclusion.

Delft University of Technology, Faculty of Applied Physics, Physics Informatics/Computational Physics, Lorentzweg 1, 2628 CJ Delft, The Netherlands.

TRAFFIC SIMULATION MODELS

Traffic simulation models or, more specifically, network assignment models have evolved from the simple, static assignment models (all-or-nothing assignment) to the more complex dynamic assignment models (three dimensional assignment model). In the past, the models have mainly been used for long- and medium-term forecasting and have played an important role in transportation development schemes. Long computations are manageable in these cases. The computation time, however, will grow rapidly with more complex models, larger networks and so on. A third important objective, short-term forecasting, has recently gained attention. Short-term forecasting is used for dynamic traffic management, which deals with on-line networkwide traffic control, route guidance, and ramp metering, for example. Short-term forecasting, in the range of 5 to 30 min, will impose a time constraint on the simulations. All together there exist enough reasons to justify a search for methods to speed up the calculations. Improvements can be made on both the algorithm side and the computer side. In the remainder of this paper possible improvements on the algorithm side will be concentrated on.

First, the collection of assignment models will be looked at to find a generalized form. Although the reader will be familiar with the models, they will be summarized.

Categorization of the Models

The simplest model is the all-or-nothing model. This model, however, does not take into account the multiple routes different travelers, from one origin-destination (O-D) pair, take in reality; nor does it take into account the load dependence of travel times or time dependence in general. The remaining models can be split into three categories, on the basis of their resolution of these three deficiencies. The corresponding algorithms are, respectively, the stochastic, the equilibrium, and the dynamic assignment algorithms. A few examples of each category will now be listed.

Stochastic Assignment

The principle of stochastic assignment is to simulate the route choice by assuming the traveler does not have perfect knowledge about his route. There are two essentially different stochastic assignment methods. The first one iteratively performs all-or-nothing assignments on the basis of randomly adjusted arc travel times. The assignment at each iteration is a fraction (1/number of iterations) of the total assignment. The maximum number of possible routes between one origin and destination is equal to the number of iterations. The second method consists of only one or two iterations but results in a large number of taken routes. Both methods have been described in detail in Van Vliet (*1*). These methods are mainly used in noncongested networks.

Equilibrium Assignment

In this category, the travel times depend on the traffic load. Multiple routes are found at equilibrium. The equilibrium is based on the principle of Wardrop (*2*): "The traveltimes of all used routes between an origin and a destination are equal to or smaller than the traveltimes of the unused routes. No traveler can shorten his journey by switching to another route." There are several ways to reach the equilibrium: methods with a theoretical background and more ad hoc methods. The practical difference is the number of iterations necessary to reach the equilibrium and the amount of work in each iteration. At the end of each iteration travel times are recalculated. Methods include

- Equilibrium: the equilibrium assignment is computed using optimization techniques. At each iteration the old assignment is compared with a new all-or-nothing assignment. The resulting assignment is the optimal weighting between these assignments. This is done until no significant improvement can be made.
- Capacity restraint: the network is iteratively loaded and unloaded, according to a predefined recipe.
- Fixed demand incremental: the network is iteratively loaded with a decreasing amount $1/(i + 1)$, where i is the iteration number.

Dynamic Assignment

Instead of observing the assignment in one time interval, assuming the total amount of traffic on a route to be evenly spread along the route, time aspects are taken into account. In other words, by dividing the observed interval into a number of periods, travelers are only contributing to the load of an arc in the period in which they are really using it. Since travelers do not start their journey all at once, the O-D matrix is defined for each period separately.

One dynamic assignment method is described by Hamerslag (*3,4*). For each period an all-or-nothing assignment is computed, resulting in an all-or-nothing assignment in three dimensions (time and space). Techniques from the previous categories are used to find a realistic assignment. The number of iterations, required in the former algorithms, is now multiplied by the number of simulated periods. This method can realistically simulate traffic in congested networks. It is possible to simulate temporary decreasing capacities caused by, for instance, accidents or ramp metering.

Summary

It can be concluded that all algorithms are repetitive all-or-nothing assignments. The workload (computational load) comes almost entirely down to the all-or-nothing assignment. To speed up the computation, optimizing the all-or-nothing assignment must be concentrated on.

The All-Or-Nothing Assignment

The all-or-nothing assignment, as the name suggests, assigns "all" traffic to a single shortest route and "non" to the others. It will, therefore, compute the shortest path between each O-D pair and assign the associated amount of traffic to each consecutive arc along that path.

```
# Compute the all-or-nothing assignment
for each O-D pair in the network {

    Compute the shortest path between origin and destination

    Assign the path

}
```

It is more efficient to calculate shortest path trees (*spt's*), because the corresponding parts of the shortest paths from one origin to several destinations can be long. Thus

```
# Compute the all-or-nothing assignment

for each origin in the network {

    Compute the spt

    Assign the spt

}
```

We will now examine the algorithm in more detail.

Shortest Path Finding Algorithms

There are four main techniques of finding shortest paths:

- Heuristic technique—one-to-one;
- Algebraic technique—all-to-all;

- Combinatorial technique—one-to-all; and
- Optimizing technique—one-to-all.

For this study, the heuristic technique is not suitable (one-to-one) and the algebraic technique is inefficient and uses much memory. This leaves the combinatorial and the reoptimizing techniques. The reoptimizing technique is the fastest technique and is based on the reoptimization of an existing *spt* for origin u to a new *spt* for origin v. As it is a more complex algorithm, consumes a large amount of memory, and is only about two times faster, one might prefer to use the combinatorial technique.

The algorithms using the combinatorial technique can be split into two groups, the label-correcting and the label-setting algorithms. The labels are the values associated with each node in the network representing the cost of traveling coming from the present origin-node (root of the *spt*).

The following algorithm is a general shortest path finding algorithm:

```
# Compute the spt
Initialize  $d_v = \infty, v \in N - \{r\}, d_r = 0$
Init_Q  $Q = \{r\}$
While ($Q \neq \phi$) {
    Select_node select $u \in Q$; $Q = Q - \{u\}$
    for each $(u,v) \in FS(u)$ {
        if ($d_u + c_{u,v} < d_v$) {
            $p_v = u$
            $d_v = d_u + c_{u,v}$
            Update_Q  $Q = Q \cup \{v\}$
        }
    }
}
```

For an explanation of the symbols used, see Appendix.

The tree is built from the root; therefore, these algorithms are also called tree-builder algorithms. The set Q contains the nodes that need to be examined. All algorithms are similar except for the way Q is maintained in **Init_Q**, **Select_node**, and **Update_Q**. The efficiency of the algorithm depends on the way this is done. The minimum computation time is reached when **Init_Q**, **Select_node**, and **Update_Q** cost minimal time.

Well-known algorithms are Moore (*5*), which is a label-correcting algorithm and Dijkstra (*6*), which is a typical label-setting algorithm. A more detailed description can be found in (*7–10*). A simple label-setting algorithm is one that simply sorts the entries in Q, (S-ord). It has a time complexity of $O(m.n)$, where m is the number of arcs in the network and n the number of nodes. The fastest algorithm using the combinatorial technique is a threshold algorithm (T-calc), which combines the good qualities of both the labeling methods. Although, in principle, the computational complexity of this algorithm is $O(n.2^n)$, and thus about the worst possible, in practice it behaves like $O(n)$ and is robust. For the maintenance of Q it uses a combination of methods: address calculation and a lifo-procedure (last-in-first-out) [see van Grol & Bakker (*11*)]. In this way, the algorithm dependent parts (**Init_Q**, **Select_node** and **Update_Q**) cost minimal time. The differences between the methods increase with growing network size. Having reduced the time complexity of the *spt* to $O(n)$ the time complexity of the assignment part will now be examined.

Assignment

The simplest way of assignment is to follow the O-D path and assign the associated amount. The following is the procedure to assign one shortest path tree (q_a, $a \in M$ contains previously assigned loads):

```
# Assign the spt for origin
for each destination in the network {
    $u$ = destination
    while $u$ is not equal to origin {
        $q_{p_u,u} = q_{p_u,u} + OD$(origin, destination)
        $u = p_u$
    }
}
```

In this way the arcs near to the origin are assigned many times. The time complexity is $O(n.l_r) = O(n.n)$, where l_r is the mean number of arcs in a path.

A more efficient method can be obtained by simultaneously assigning several O-D elements to an arc. Supposing that l_u is the level in the tree (the number of nodes counted from the origin), all nodes can be sorted according to their level. Starting at the highest level, the following procedure can be executed.

```
# Assign the spt for origin
K = $\{1 \ldots n\}$, $S_u = 0, u \in N$
# Sort the nodes in set $K$ top-level down
Sort $K$
for each node $u$ from $K$ {
    $q_{p_u,u} = q_{p_u,u} + OD(\text{origin},u) + S_u$
    $S_{p_u} = S_{p_u} + OD(\text{origin},u) + S_u$
}
```

The set K contains all nodes with their levels. This reduces the complexity of the assignment to $O(n)$ but sorting K has a complexity of $O(n.\log n)$. By using an addressable array, $L()$, to sort the nodes by their level we can reduce the total complexity to $O(n)$. The algorithm then becomes

```
# Sort levels of spt for origin
$L(.) = 0$
for each node $u$ {
    $L(l_u) = L(l_u) \cup \{u\}$
}
# Assign the spt for origin
$S_u = 0, u \in N$
for each level top down {
    for each node $u$ in the set $L$(level){
        $q_{p_u,u} = q_{p_u,u} + OD(\text{origin},u) + S_u$
        $S_{p_u} = S_{p_u} + OD(\text{origin},u) + S_u$
    }
}
```

Summary

A complete description of the algorithm can be found in the Appendix. The overall computational complexity (path finding and assignment) for one *spt* is now $O(n)$. This means that a number of operations is executed on each node and $O(n)$ is thus the minimum complexity. Only the number of operations can now be minimized. The computational complexity of the all-or-nothing assignment is $O(n^2)$. A good implementation of this algorithm is the best that can be done. A major improvement could only be achieved by an implementation in assembler.

Two observations can be made. First, it can be seen that the computations of the *spt*'s are independent. This allows the use of parallelism. The second observation is that the all-or-nothing assignment is mainly a data flow problem. All network data will flow through the algorithm several times, whereas the number of operations on the data is minimal. Bearing this in mind, the possible use of special-purpose hardware is discussed in the next section.

SPECIAL-PURPOSE HARDWARE

First, the use of general-purpose computers in traffic simulations will be discussed, because these limitations motivated the current research into using special-purpose hardware. The principles of the special-purpose computers are described next. Third, dedicated architectures for traffic simulations will be focused on. After an introduction to the proposed LPA, a more detailed description of the LPA, dedicated for traffic simulations, will conclude the section.

General-Purpose Computers

Standard traffic simulations usually run on general-purpose computers, such as microcomputers, workstations, and mainframes. For small networks and simple algorithms, the turnaround time of the simulations is satisfactory for most applications. Moving from microcomputer to mainframe or minisuper workstations significantly improves performance and visualization, but larger networks and more complex assignment algorithms require supercomputer power. However, the cost and the limited availability of supercomputers eliminate this option.

In general, commercially available computers were designed to solve all problems, and are not tailored to efficiently solve a typical problem such as found in traffic simulations. To improve the cost to performance ration, or to bypass hardware limitations of general-purpose computers, one can design and build a special computer, the architecture of which maps perfectly on the problem or algorithm involved. This approach can be considered a low-cost alternative to supercomputers.

Special-Purpose Computers

Special-purpose computers are designed to efficiently carry out a particular task at supercomputer speed. In general, they cannot handle any other task, or, if they can, the performance will be poor. However, a design can cover a class of problems, and thus can be used for a wide range of applications without performance penalty. The special-purpose computer is designed on the basis of the problem, problems, algorithm, or algorithms to be used, and allows an architecture that explores parallel and pipelined operations wherever applicable. It allows problem-dependent memory organization, problem-adapted basic instruction set, and so on with the purpose to improve the total performance of the computer.

Special-purpose computers range from single-purpose to multi-purpose computers and they differ in the flexibility of programming them.

- *Single-purpose computers.* Single-purpose computers have a basic instruction set that will only cover the operations required for the task it has been designed for. This approach allows the algorithm to be hardwired, which guarantees an optimal speed. Fixed-wired parallel and pipelined architectures restrict the flexibility to modify the algorithm, but leave open the possibility to vary enough parameters to motivate the effort to design such a machine. The cost to performance ratio of this type of computer is low (factor 100 better than supercomputers) and they are available 24 hr/day for the computer experimentalist. A variety of single-purpose computers have been successfully exploited in signal processing and computational physics (*11–14*).
- *Multi-purpose computers.* Multi-purpose computers are designed to efficiently solve a class of problems rather than a single problem. These architectures reflect the common

property of the algorithms involved and can be programmed to solve a problem from the class of problems it was designed for. High-level languages are used (C and F77) to program the computer. The speed is obtained by using many processor nodes interconnected by a communication network that is suitable for the class of algorithms involved. The architecture of the nodes is kept simple but effective to allow the construction of efficient compilers. The choice of processor, memory structure and size, interconnection network, and word length characterize the multi-purpose computer. They can be shared memory or distributed memory machines running in single-instruction multiple-data or multiple-instruction multiple-data mode. Flexibility and programmability of these computers are traded for ultra speed as in single-purpose computers. However, the newest commercially available microprocessors are fast, the architecture is scalable, and can result in a cost to performance ratio improvement by two orders of magnitude compared with supercomputers.

So-called general-purpose parallel computers, such as N-cubes, may use fast processor nodes, but their memory architectures and their slow interconnection networks do decrease the overall performance dramatically. Only between 10 and 20 percent of the advertised peak performance is reached by careful programming. Existing parallelizing high-level language compilers are still far from ideal. Automatic decomposition of sequential program flows of problems that are often parallel in nature is not an efficient way to obtain fast codes for parallel computers.

In practice, the user has to choose a network topology and program the nodes to use that network efficiently. Often topologies are chosen to be ring structures, so that the programmer can implement the algorithm without begin distracted by more exotic topologies. Still, the node interconnections are slow because of their all topology structure.

A Special-Purpose Computer For Traffic Simulations

To select or design a computer for traffic simulations, the algorithm has to be examined for possible parallel or pipelined operations. As the algorithms involved are still in development, thus demanding flexibility, a single-purpose computer will not be considered. Decomposition of the total problem into coupled parallel processes is a way to find a scalable parallel architecture. The node architecture, memory size, and interconnection channel will then determine the final computer.

As shown previously, a time critical part of the algorithms used in traffic simulations is the calculation of n shortest path trees (*spt*'s), where n is the number of nodes in the network. It was concluded that the *spt*'s can be independently calculated, thus allowing us to decompose the problem into n problems of one *spt*. Using n processor boards, all the *spt*'s can be calculated in parallel. This will improve the simulation speed by order n. In a parallel computer, with P processor boards, n/P *spt*'s can be calculated on each board, which gives a speed-up factor of P. The latter solution is preferable for reasons of scalability, especially for large n. Each processor board will need all network information, thus P times the amount of memory needed to store the network is the minimal total memory size. Minimizing P to keep memory costs down is compensated by making the processor boards as fast as possible, and keeps the overall performance high. In the preceding assignment schemes, this decomposition also holds. However, to find the total load per arc, the partial arc loads must be accumulated. It is necessary, therefore, to efficiently interconnect the processor boards. Here, a pipeline structure in which each processor board is one pipe stage of the whole pipe can be used. Consequently, the processor boards are ordered in one string to construct this pipeline. One high-speed data channel from each board to its neighbor board is sufficient to obtain a fast pipeline. The accumulated arc loads are collected in the last pipe stage (last processor board in the string). To start the next iteration, the updated network has to be broadcast to all processor boards. When all processors are connected to a common bus, broadcasting can be accomplished using one talker and P listeners on this bus (see Figure 1). Before going into details of the design, a more generalized architecture that will cover the above architecture ideas for traffic routing simulations but can be used for other purposes as well (multi-purpose computer) is discussed.

LPA Architecture

An LPA is a one-dimensional array of identical processor boards, each of which is connected to its two neighboring boards only by a data bus. In addition, the boards share a common data-, address-, and control-bus, which is also interfaced to a general-purpose host computer. One processor board is configured as a control board, which supervises the chain of processor boards through a special control bus (see Figure 2).

A large class of problems in computational physics (both authors work in this field) can be solved using this parallel architecture. A natural domain decomposition allows the mapping of different subdomains on different processors. All subdomains can be processed simultaneously. In general, the calculations in a subdomain need data from the other subdomains, but when "local environment problems" are

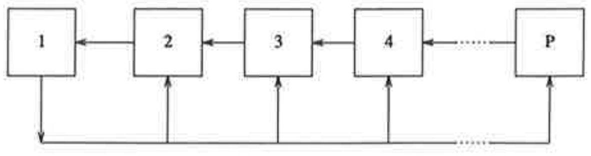

FIGURE 1 Data flow in the traffic assignment problem mapped on an array of *P* processor boards.

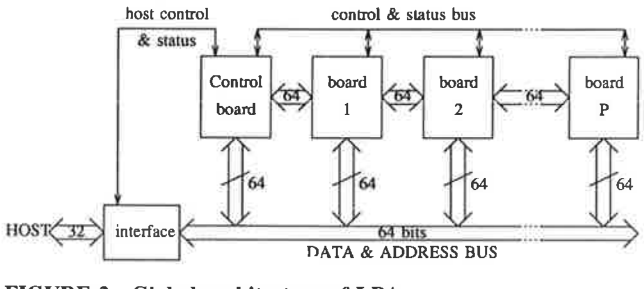

FIGURE 2 Global architecture of LPA.

involved, such as in finite difference calculations, only the directly neighboring subdomains are contributing to the results. For this class of problems, the domain can be decomposed by slicing in just one direction. Each domain slice is mapped onto one processor, and each processor communicates only with its two neighboring processors. The host passes data and programs to the LPA and collects results of the calculations by means of the common bus. The processor architecture, the local memory and the interface to the two neighboring processor boards are optimized to tackle the problem they are designed for.

LPA for Traffic Simulations

LPA architecture is well suited for the traffic-routing problem. Programs and network data, which are (mostly) identical on the different boards, can be broadcasted to the processor boards. Each processor node requires enough memory to contain all network information and the locally calculated results. At the start of every iteration (all-or-nothing assignment), the network information on all the processor nodes is identical. Then, every node starts to calculate its share of the total number of shortest paths instructed by the control board. Clearly, this calculation is intensive. A fast processor is needed for all floating point calculations. In addition, because a lot of data are involved, the memory interface is important. Bus contention should be avoided at all times. This is partly solved by the parallel approach taken, with independent nodes and distributed memory, but a fast memory interface tuned for this particular problem is still called for.

The processor needs to be selected primarily on grounds of floating point calculation speed. The fastest processor available today is the Intel i860 Microprocessor. It contains a core unit, floating point unit, and instruction and data caches on one chip. Theoretical speeds are 40 million instructions per sec (MIPS) and 80 million floating-point operations per sec (MFLOPs). Practical speeds using high-level languages such as C and Fortran are in the order of 15 MFLOPS. Hand coded assembly is capable of performing between 30 and 80 MFLOPS, depending on the algorithm, and the amount of vectorization and pipelining that can be employed. Furthermore, the Intel i860 can execute integer and floating point operations in parallel and has a 64-bit-wide memory bus.

Because of the speed at which the processor processes data, the memory interface has become the only bottleneck. The rate at which data can be retrieved from and stored into memory determines the overall processing speed. The network information is typically scattered through memory, thus rendering the on-chip (small) data cache practically useless. Therefore, the interface to the dynamic memory (use of static memory only would be too expensive) is vital.

At the end of each iteration, every board has calculated part of the load for every arc. These partial loads can be accumulated in a pipelined fashion using the connections to the neighbor boards: every board receives the partial loads from its right neighbor, adds its own partial loads, and hands the results to its left neighbor. Finally, the accumulated partial loads are handed to the control board (which is the left-most board), which can start the next iteration by broadcasting the updated network.

The connections between the boards are realized by first-in first-out buffers (FIFOs), which are 64-bit-wide memory components that move data in receiving order to the neighbor board (size of buffer is several kByte). The FIFOs are used to automatically synchronize the asynchronous processors, and buffer data sent between them. This allows the processor nodes to act autonomously and send data whenever ready. The control board is not needed to synchronize nodes or buffer data. All processor nodes can use their FIFOs concurrently, allowing for maximum throughput.

RESULTS

It was previously shown that the performance of traffic simulation programs can be improved and how special-purpose hardware can be used to reduce the computing time and the cost involved. The improvements expected by using special-purpose hardware will now be defined.

First, the improvement by parallelization will be considered. Using an architecture by the LPA concept is proposed, with 16 i860-based processor boards, an improvement of a factor 16 compared with one i860-based processor board can be expected. Although this seems obvious, most computers with parallel processors are unable to improve their performance by the number of processors they use—compared with one processor [see (*15,16*)]. The improvement is justified by the negligible overhead in interboard communication. The interboard communication, required to accumulate the arc loads and to broadcast network data, does not increase with the number of processor boards and is small compared to the total computation.

An operating system running on a computer allows the users to use all kinds of facilities, such as file-support, I/O in general, multitasking, scheduling, timing and so on. Using such an operating system will decrease the overall performance of the system. By avoiding most of these facilities some performance improvement can be gained. The operating systems running on the LPA nodes will allow the minimum amount of facilities to run the problem efficiently.

Second, the expected performance of the LPA with some general-purpose computers that are commonly used are compared. The i860 is a fast processor as explained previously. The processor alone is already in competition with several fast general-purpose computers. Some test runs have been executed on the Intel Microprocessor Software Development system—STAR860—which is an AT-386 with an i860 CPU based add-on card. This board is not optimally configured, and can thus be improved on the performance in the final design.

Comparison tests have been run on a Convex C1, several differently configured Silicon Graphics (SG, R3000, 33-25-20 MHz, 64k-Byte cache), a MicroVAXIII, a MicroDutch (68020), an Hewlett Packard Workstation (HP, 68030), and a personal computer (AT-386, 25Mhz, 64kByte cache). The Convex C1 is a vector processor with an architecture resembling a Cray supercomputer. Although not the fastest, it is a widely used computer. The test programs were written in C. Two shortest path tree algorithms discussed previously, S-ord and T-calc, were used. The assignment was done in two different ways, the 'simple' assignment and using levels, also described previously. For the implementation of the algorithms we can use pointers or indexing. The calculations can be done in integers or in floating point notation. The resulting computing times are given in Tables 1–3.

On all computers available optimizers were used. The network used contained 3,347 nodes, 9,394 arcs, and an arc to node ratio 2.8. As comparison, a network of 17,931 nodes was also used. The computing times given are the all-to-all times; an all-or-nothing assignment with each node as origin and destination. The i860 is not the fastest processor in Table 1 because of its memory configuration. The Silicon Graphics (33MHz) has an advantage of having a 64kByte cache. This advantage disappears when a larger network is used (see Table 3). With integer indexing the i860 is always superior. Next to an upgrade of the i860 from 33 MHz to 40 MHz, the processor board can be improved by using a pipelined-multibank memory instead of the single-bank memory implementation on the STAR860.

CONCLUSIONS

The use of special-purpose hardware is only legitimate when the task to be performed is time critical and the budget is limited. It was shown that the STAR860 is two times as fast as the Convex C1. The performance of a single i860-based processor board, in comparison to the STAR860, can be improved by a factor of about 2, using a faster version of the i860 and a better memory architecture.

The price of a single i860-based processor board is mainly determined by the memory cost and, depending on the amount of memory, is estimated to be between $5,000 and $10,000. A 16-board LPA together with a general purpose host, of about $40,000, and additional costs of manufacturing of about $15,000 amounts to a total cost of less than $0.3 million. The price of a Convex C1, however, is about $0.6 million, and thus leads to another factor 2 in cost to performance improvement. Hence, a 16-board LPA will give a cost to performance improvement of a factor 100 compared to a Convex C1. For future developments of on-line traffic control, route-guidance, ramp metering and so on, the LPA should be a powerful tool.

TABLE 2 THE DIFFERENCE IN COMPUTING TIMES BETWEEN USING A SIMPLE ASSIGNMENT METHOD AND ONE USING LEVELS

Data-struct:	Pointer			
spt-alg	T-calc		S-ord	
Assignment	Levels	No-levels	Levels	No-levels
SG (20 MHz)	266	703	315	751
SG (25 MHz)	260	665	302	701
SG (33 MHz)	161	438	189	464
STAR860	190	535	202	548

NOTE: The times are given for two *spt* algorithms and on several computers. The network size $N = 3{,}347$. SG is Silicon Graphics.

TABLE 3 THE DIFFERENCE IN PERFORMANCE OF THE COMPUTERS ON NETWORKS WITH DIFFERENT NETWORK SIZES

Size	3,347	17,931
Convex C1	411	683
SG (25 MHz)	260	581
SG (33 MHz)	161	417
STAR860	190	319

NOTE: The times given are in case of $N = 3{,}347$, all-to-all, in case of $N = 17{,}931$, 1,000 *spt*'s. The *spt*-algorithm used was T.calc and the assignment uses levels. SG is Silicon Graphics.

ALL-OR-NOTHING ASSIGNMENT

The following is a description of the algorithm used to perform the all-or-nothing assignment. This is an optimal algorithm depending on the way **Init_Q**, **Select_node**, and **Update_Q** are implemented. The symbols used are defined as follows:

N, M	=	set of all nodes, arcs in the network;
n, m	=	number of nodes, arcs in the network;
$FS(v)$	=	the foreward star representation of node v, defines the network;
$OD(r,v)$	=	matrix, number of travelers going from node r to v;

TABLE 1 COMPUTING TIMES IN SECONDS ON A NUMBER OF COMPUTERS

	Pointers				Indexing			
Data-struct: Notation:	Floating Point		Integer		Floating Point		Integer	
spt-alg	T-calc	S-ord	T-calc	S-ord	T-calc	S-ord	T-calc	S-ord
Microdutch	4,307	6,155	2,626	2,989	6,372	10,284	4,352	6,966
AT-386	1,485	2,137	733	674	1,807	2,823	840	1,305
MicroVAXIII	1,180	1,712	1,119	1,324	1,830	3,004	1,635	2,585
HP	1,149	1,562	789	764	1,674	2,683	1,133	1,747
Convex C1	411	535	315	382	515	744	384	565
SG (20 MHz)	266	315	237	273	268	365	230	341
SG (25 MHz)	260	302	238	270	319	407	276	393
SG (33 MHz)	161	189	139	164	222	286	187	268
STAR860	190	202	185	192	235	283	179	240

NOTE: The program, either T-calc or S-ord, is implemented with pointers or indexing and the calculations in either integer or floating point notation. The assignment is implemented with the use of levels. The network size $N = 3{,}347$. HP is Hewlett Packard, SG is Silicon Graphics.

c_a, $c_{u,v}$	=	used arc length (arc travel time) for arc a from node u to node v;
d_v	=	calculated distance (travel time) from the origin node to node v;
p_v	=	previous node from node v in the shortest path tree (*spt*), defines the *spt*;
l_v	=	level of node v in the shortest path tree;
q_a, $q_{u,v}$	=	calculated load on the arc a, from node u to node v;
r	=	current origin-node (root);
Q	=	temporary set, contains nodes to be examined;
$L(n)$	=	set containing all nodes from *spt* on level n; and
S_v	=	temporary, traffic load going up to node v.

In short, the all-or-nothing assignment looks as follows:

```
# Compute the all-or-nothing assignment

for each origin in the network {

    Compute the shortest path tree

    Sort levels

    Assign the spt

}
```

The subroutines **Init_Q**, **Select_node**, and **Update_Q** are not defined here. Although crucial to the efficiency of the algorithm, the functionality remains the same. The calculated loads from preceding assignments are kept in q_a, $a \in M$.

```
# Compute the shortest path tree for origin $r$

Initialize    $d_v = \infty, v \in N - \{r\}, d_r = 0, l_r = 1$

Init_Q    $Q = \{r\}$

While ($Q \neq \phi$) {

    Select_node select $u \in Q$; $Q = Q - \{u\}$

    for each $(u,v) \in FS(u)$ {

        if ($d_u + c_{u,v} < d_v$) {

            $p_v = u$

            $l_v = l_u + 1$

            $d_v = d_u + c_{u,v}$

            Update_Q    $Q = Q \cup \{v\}$

        }

    }

}
```

```
# Sort levels

Initialize    $L(.) = 0$

for each node u {

    $L(l_u) = L(l_u) \cup \{u\}$

}

# Assign the shortest path tree

Initialize    $S_u = 0, u \in N$

for each level top down {

    for each node u in the set $L(level)$ {

        $q_{pu,u} = q_{pu,u} + OD(r,u) + S_u$

        $S_{pu} = S_{pu} + OD(r,u) + S_u$

    }

}
```

REFERENCES

1. D. Van Vliet. Road Assignment I, II en III, *Transportation Research*, Volume 10, 1976, pp. 137–157.
2. J. G. Wardrop. Some Theoretical Aspects of Road Traffic Research. *Proc. of the Institute of Civil Engineers*, Part II, 1, 1952, pp. 325–581.
3. R. Hamerslag and P. C. H. Opstal. *A Three-Dimensional Assignment Method in Time-space.* Report 87-49 from the Faculty of Mathematics and Informatics, Delft University of Technology, The Netherlands, 1987.
4. R. Hamerslag. Dynamic Assignment in the Three Dimensional Timespace. In *Transportation Research Record 1220*, 1989.
5. E. F. Moore. The Shortest Path Through a Maze. *Proc. of International Symposium on the Theory of Switching*, Harvard University Press, Cambridge, MA, 1959, pp. 285–292.
6. E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1, 1959, pp. 269–271.
7. R. Dial, F. Glover, D. Karney, and D. Klingman. A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees, *Networks 9*, 1979, pp. 215–248.
8. G. Gallo and S. Pallottino. Shortest Path Methods: A Unifying Approach. *Mathematical Programming Study*, 26, 1986, pp. 38–64.
9. F. Glover et al. *Threshold Assignment Algorithm.* Report CBDA 107, Center for Business Decision Analysis, University of Texas, Austin, 1982.
10. H. J. M. van Grol and A. F. Bakker. Shortest Path Finding: From algorithm to Special Purpose Hardware. *Proc., Transportation Planning 17th PTRC Transport & Planning Summer Annual Meeting*, University of Sussex, England, 1989.
11. A. F. Bakker and C. Bruin. Design and Implementation of the Delft Molecular-Dynamics Processor. In *Special Purpose Computers* (Berni J. Alder, ed.), 1988, Chapter 6. ISBN 0-12-049260-1
12. D. J. Auerbach, A. F. Bakker, T. C. Chen, A. A. Munshi, and W. J. Paul. A Highly Parallel Computer for Molecular Dynamics Simulations. *Materials Research Society Symposium Proc.*, Vol. 63, 1985.

13. D. J. Auerbach, W. J. Paul, A. F. Bakker, C. Lutz, W. E. Rudge, and Farid F. Abraham. A Special Purpose Parallel Computer for Molecular Dynamics: Motivation, Design, Implementation, and Application. *Journal of Physical Chemistry*, Vol. 91, 4881, 1987.
14. A. F. Bakker, G. H. Gilmer, M. H. Grabow, and K. Thompson. A Special Purpose Computer for Molecular Dynamics Calculations. *Journal of Computational Physics*, Vol. 90, No. 2, 1990.
15. Huey-Kuo Chen and David E. Boyce, Code Optimization For A Nonlinear Transportation Network Model: A Case Study, *Presented at the 68th Annual Meeting of the Transportation Research Board*, Washington, D.C., 1989.
16. K. C. Mouskos and Hani S. Mahmassani. Guidelines and Computational Results for Vector Processing of Network Assignment Codes on Supercomputers. *Presented at the 68th Annual Meeting of the Transportation Research Board*, Washington, D.C., 1989.

Publication of this paper sponsored by Committee on Transportation Supply Analysis.