Dynamic Traffic Assignment on Parallel Computers

Marcus Rickert and Kai Nagel*
Los Alamos National Laboratory and Santa Fe Institute
1399 Hyde Park Rd, Santa Fe NM 87501, U.S.A.

May 10, 1999

Abstract

We describe part of the current framework of the TRANSIMS traffic research project at the Los Alamos National Laboratory. It includes parallel implementations of a route planner and a microscopic traffic simulation model. We present performance figures and results of an offline load-balancing scheme used in one of the iterative re-planning runs required for dynamic route assignment.

1 Introduction

In the context of urban planning, it would be useful to have a computational tool that evaluates transportation consequences of urban evolution scenarios. The anticipated urban structure, including anticipated demographic data and anticipated transportation infrastructure, could then be fed into a computer, and the computer would calculate the resulting traffic. Maybe the tool would even have a virtual reality component, enabling one to zoom to one's home and count the number of passing cars during the rush hour (see Fig. 1). Clearly, such a tool should include the effects of "induced" traffic, i.e. the observation that lower congestion levels encourage people to more travel; the tool should be able to say something about the variability of traffic; and it should be capable of incorporating new technology, such as telecommuting or telematics systems.

The problem of induced traffic makes clear that transportation is *not* a simple infrastructure problem, where to find a good or optimal solution to move a given demand, but a problem where individual people's values and preferences play an enormously important role. In consequence, any quantitative technique needs to be able to represent aspects of human decision-making. The problem thus becomes as much a problem of (computational) social science as one of engineering.

The TRANSIMS (TRansportation ANalysis and SIMulation System) project at Los Alamos National Laboratory [1] is an attempt to build such a tool. The key to the TRANSIMS design is that it is completely microscopic, which means that it keeps track of individual travelers throughout its modules. Similarly, elements of the transportation infrastructure, such as intersections, traffic lights, turn pockets, etc., are represented microscopically.

In TRANSIMS, each traveler is a computational agent. Agents make *plans* about what to do during a day – in order to get from one activity location to another, agents can, for example, walk, use bicycles, drive cars, or use busses.

*Corresponding author. Email kai@santafe.edu

Eventually, all plans are simultaneously executed in a micro-simulation of the transportation system.

In principle, this leads to a straightforward simulation approach (see Fig. 2): Derive synthetic households from demographic data and locate them on the network; use the demographic information together with land use information to derive activities (working, sleeping, eating, shopping, etc.) and activities locations for each household member; and let agents decide about mode and routing for their transportation. So far, all these are *plans*, i.e. *intentions* of the simulated individuals. These plans can then all be fed into a realistic transportation micro-simulation, which can be used as the basis for analysis, such as emissions calculations.

The advantage of such a microscopic approach is that, at least conceptually, it can be made arbitrarily realistic. This makes it possible to include dynamic effects such as queue spillover, which are sometimes hard to represent in traditional methods. It also makes it possible to include new and perhaps unanticipated technology at a later time. For example, the whole architecture of ITS (Intelligent Transportation Systems) can be mirrored by a careful software implementation.

Yet, there are also several disadvantages, some of them being:

(i) Size of the problem: Metropolitan regions typically consist of several millions of travelers. Executing a second-by-second transportation micro-simulation on a problem this size within reasonable computing time is only possible with the use of advanced statistical and computational techniques.

(ii) Behavioral foundation: We are far from understanding human behavior. For that reason alone, we are unable to predict the behavior of *individual* travelers. However, there seems to be a realistic chance that the *macroscopic* (emergent) behavior that is generated by thousands or ten-thousands of interacting individuals is considerably more robust than the behavior of an individual agent. This would be similar to Statistical Physics, where the trajectory of a single particle is unpredictable, yet, useful macroscopic properties of gases such as equations of state can still be derived.

(iii) Consistency problem: The approach outlined above is not as straightforward as it sounds because the plans depend on *expectations* about traffic conditions during execution. For example, if a person expects congestion, he or she may make different plans than when no congestion is expected. Yet, congestion occurs only when plans *interact* during their simultaneous execution. In short, plans depend on congestion, but congestion depends on plans. – This logical deadlock is not unknown in economic theory and is traditionally overcome by the assumption of rational agents. Both with and without the assumption of rationality, this problem of consistency between plans and micro-simulation makes the computational challenge even bigger.

(iv) Robustness: Any approach to a problem needs to have reproducibility of the results under a wide enough range of changes, or otherwise the results are useless for practical purposes.

This paper concentrates on the computational problems, and what we have done to solve them.

2 Dynamic traffic assignment

First let us briefly describe the traditional approach to the problem. The traditional urban transportation planning process consists of four steps [2]:

- Trip generation: Flows out of and into so-called traffic analysis zones are generated.
- Trip distribution: After the trip generation, we only have sources and sinks of traffic, but not how they are matched. This is done in the trip distribution part. As a result, one now has an origin-destination table, i.e a matrix with origins as columns and destinations as rows, and each entry denotes the amount of traffic from the respective origin to the respective destination.
- Modal split: The traffic streams are distributed on the different modes.
- Assignment: The traffic streams are assigned on particular routes on the network.

TRANSIMS, in contrast, generates activities, that is, origin and destination information is always attached to the people. Nevertheless, TRANSIMS also at a certain point has all the origin-destination relations given and needs to assign these on a network. One could even translate the TRANSIMS information to origin-destination matrices, although one would give information away, such as exact starting times of trips, and information from trip chaining, e.g. the effect that congestion in the morning may lead to later departure times in the evening.

Traditional assignment [2] assumes that traffic streams are constant throughout the evaluation period. Often, only the morning or afternoon peak is selected as such an evaluation period. However, when the time periods become too short, the theoretical foundation of these models is no longer valid. Yet, longer time periods (such as an hour) are not capable of generating dynamic short-term effects, such as queue build-up or congestion spreading during the onset of the rush hours.

Iterated microsimulations of traffic provide an alternative for the assignment portion. Several groups (e.g. [3, 4, 5, 6, 7, 8]) have used the iterative approach of *routing–microsimulation–feedback of travel-times* to obtain an assignment (route set) that is, within the accuracy permitted by any implicit or explicit stochasticity of the model, self-consistent. In this paper we outline part of the framework of TRANSIMS [1, 9] as it was used for an extensive case-study of the Dallas / Fort Worth (Texas) street network, and for preliminary studies using data from the Portland (Oregon) metropolitan area.

The "Dallas" study [9] used origin-destination matrices provided by the local transportation planning authority (NCTCOG) as input. It used a so-called focussed road network, which means that for a 5 miles times 5 miles study area *all* streets were represented, whereas with further distance from the study area more and more of the less-important streets were left out. This network countained 9864 nodes and 24 622 uni-directional links, of which 2276 nodes and 6124 links were in the study area.

The Portland study is planned to be run on the complete road network, including all local streets. This network contains about 200 000 uni-directional links. We also use another network, with 20 000 links, which has been used by the local transportation planning authority (Portland Metro) for their traditional assignment studies. Demand generation will this time be achieved via activities generation, as intended by the TRANSIMS design. Results in this paper are based on very preliminary sets of home-to-work trips [10]. This should be of no consequence for the computational results presented in this paper.

The simulation set-up itself consists of two main applications: (a) a route planner based upon a fast implementation of the Dijkstra algorithm that uses time-

dependent link travel-times to compute shortest routes, and (b) a traffic microsimulation that executes the routes generated by the planner and supplies a feedback file which is used in subsequent calls of the router.

In the following sections we will describe the two applications and their mutual dependency in more detail.

3 Application Framework

As stated above, the basic input of the dynamic assignment process is a list of trips, defining what vehicles depart from which *origin* at what *departure time* to what *destination*. For all considerations presented in this paper, the list of trips is regarded as given and fixed. The main goal lies in assigning actual route plans to these trips which fulfill a certain optimization criterion, e.g. minimizing each individual's travel-time based upon the actual time-dependent link travel-times which would be generated by executing the route-plans. In TRANSIMS a microsimulation is used to provide (and verify) the link travel-times generated by the route-plan. The goal is to generate route plans which are (within the limitation of the stochasticity of the process) optimal.

Now, initially, the algorithm cannot predict which route will be the fastest, since that depends on congestion, and no information about congestion is available initially. This problem is solved via an iterative relaxation approach, that is, one generates an initial set of routes, runs it through the micro-simulation, re-plans a fraction f_r (or all) of the trips, runs the micro-simulation again, etc., until some convergence criterion is fulfilled. Choosing f_r too large usually results in oscillations (e.g. [11]), except when other measures are taken such as giving different routes different weights and to distribute trips across the routes according to those weights (e.g. [3]). Figure 3 depicts the data flow during an iteration series. Figure 4 shows the accumulated travel-time as a function of the iteration number for different iteration strategies and re-planning fractions of 1% and 5% (see [12]). Approximately 40 to 60 iterations with 5% re-planning are required before the travel-time has reached a sufficiently relaxed state.

4 Route Planner

During the initial iteration the route planner reads the trip file and a (potentially time-dependent) link travel-times file to generate a route set. For each subsequent step, it uses the route set of the previous step (which includes the trips implicitly) and re-routes a fraction of all trips. The travel-time feedback from the microsimulation is averaged over bins of width $t_b = 900$ seconds. Both for the initial route planning and for re-planning, each route is handled independently, i.e. the planner itself does not keep track of the delays that will be caused by travelers on the links they are intending to use. The algorithm used is a fast implementation of a time-dependent Dijkstra [13].

In order to obtain more computational speed, the route planner uses a classical master-slave parallelization with PVM as message passing library. The master reads in the current route-plan and distributes trips by coding them into messages and sending them to the slaves. Each slave reads a copy of the travel-time feedback file for its shortest path computations. The slaves return routes which are sorted with respect to departure time and written to a new route set file by the master. Since the computations of the routes are completely independent, there is no communication between the slaves at all.

Figure 5 shows the execution times for processing 100,000 plans at two different re-planning fractions. The data refers to the 14751 link network used for Dallas, run on a SUN Enterprise 4000 with 14 CPUs running at 250 MHz. The speedup is better for larger re-planning fractions f_r because the ratio between shortest path computations and I/O overhead improves. Still, current speed-up is not very satisfactory and we may have to think about more efficient methods such as packaging several route requests into single messages, or maybe even completely abandoning the master-slave approach.

5 Microsimulation

5.1 Driving Logic

The microsimulation uses a cellular automata (CA) technique for representing driving dynamics (e.g. [14, 15]). The road is divided in cells, each of a length that a car uses up in a jam (we currently use 7.5 meters). A cell is either empty, or occupied by exactly one car. Movement takes place by *hopping* from one cell to another; different speeds are represented by different jumping distances. Using one second as the time step works well (because of reaction-time arguments [16]); this implies for example that a hopping speed of 5 cells per time step corresponds to 135 km/h. As long as no lane changing takes place, this just models "car following"; the rules for car following in the CA are: (i) linear acceleration up to maximum speed if no car is ahead; (ii) if a car is ahead, then adjust velocity so that it is proportional to the distance between the cars; (iii) sometimes be randomly slower than what would result from (i) and (ii).

Lane changing is done as pure sideways movement in an additional "half time-step" before the forwards movement of the vehicles, i.e. each time-step is subdivided into two sub-time-steps. The first sub-time-step is used for lane changing, while the second sub-time-step is used for forward motion. Each sub-time-step requires the exchange of boundary information between CPUs. Lane-changing rules for TRANISMS are symmetrical and consist of two simple elements: Check if the other lane is faster, and if there is enough space to "get in" [17]. Two other important elements of traffic simulations are signalized turns and so-called unprotected turns. The first of those can for example be modelled by putting a "virtual" vehicle of maximum velocity zero on the lane when the traffic light is red, and to remove it when it is green. Unprotected turns get modelled via "gap acceptance": There needs to be a certain gap on the lane which has the priority so that the car from the minor road accepts it. For further information, see, e.g. [18].

5.2 Micro-simulation parallelization

The main advantage of the CA is that it forces the design of a parallel and local update, that is, the state at time step $t + 1$ depends only on information from time step t, and only from neighboring cells. (To be completely correct, one would have to take our sub-time-steps.) This means that domain decomposition for parallelization is straightforward (see below), since one can communicate the boundaries for time step t, then locally on each CPU perform the update from t to $t + 1$, and then exchange boundary information again. It would even be possible to overlap communication and computation, although experiments with it have not shown any systematic improvement on the machines we use.

Rickert, Wagner and Gawron [19, 20] implemented a parallelized traffic simu-

lation running several times faster then real–time[1] for the German Autobahn network on an SGI Power Challenger. A summary about how the traffic CA has been used in simulation models can for example be found in [21].

As stated above, the inherent structure of a traffic microsimulation favors a *domain composition* as the general approach to parallelization:

- The street network can easily partitioned into tiles of equal or almost equal size. A realistic measure for size is the accumulated length of all streets associated with a tile.
- The range of interdependencies between network elements are restricted to the interaction range of the CA model. All current rule sets have a interaction range of 35 meters which is still small with respect to the average link length. Therefore, the network easily separates into independent components.

We decided to have the boundaries between CPUs in the middle of links rather than at nodes. This separates the dynamical complexity at the nodes from the complexity caused by the parallelization and makes optimization easier. For example, when having the boundaries at nodes, it can easily happen that three CPUs are involved in movements across the intersection, whereas for links at most 2 CPUs are involved in any movement of a single vehicle.

In order to distribute the graph across CPUs, one approach is orthogonal recursive bi-section. In our case, since we cut in the middle of links, the first step is to accumulate computational loads at the nodes: Each node gets a weight corresponding to the computational load of all of its attached half-links. Nodes are located at their geographical coordinates. Then, a vertical straight line is searched so that roughly half of the computational load is on its right and the other half on its left. Then the larger of the two pieces is picked and cut again, this time by a horizontal line. This is recursively done until as many tiles are obtained as there are CPUs available, see Fig. 6. It is immediately clear that under normal circumstances this will be most efficient for a number of CPUs that is a power of two. As one consequence, the tiles can directly exchange boundary information containing all data necessary for the evaluation of the CA rule sets, resulting only in local communication between neighboring tiles.

Another option is to use the METIS library for graph partitioning [22]. That library considerably reduces the number of boundary edges, as shown in Fig. 7. And example of the resulting tiling can be seen in Fig. 8; for example, the algorithm now picks up the fact that cutting along the rivers should be of advantage.

Such an investigation also allows to compute the theoretical efficiency based on the graph partitioning. Efficiency is optimal if each CPU gets exactly the same computational load. However, since for the entities that we distribute the weights are given, load imbalances are unavoidable, and they become larger with more CPUs. We define this theoretical efficiency due to the graph partitioning as

$$eff := \frac{\text{load on optimal partition}}{\text{load on largest partition}} , \qquad (1)$$

where the load on the optimal partition is just the total load divided by the number of CPUs. We then calculated this number for actual partitionings of both of our 20 000 links and of our 200 000 links Portland networks, see Fig. 9.

[1] Running at several times real-time means that several simulation seconds can be computed in one wall-clock second.

The result means in short that, according to this measure, our 20 00 links network would still run efficiently on 128 CPUs, and our 200 000 links network would run efficiently on up to 1024 CPUs.

The simulation uses a parallel update with a global time-step. However, synchronization of all CPUs is only performed after a *simulation sequence* comprising approximately 10-20 time–steps. In between, there is only an implicit synchronization through the exchange of local boundaries.

The global time–step is used to guarantee consistent collection of statistical data: Although partial results from the CPUs may not be collected at the same physical wall-clock time due to a potential time-step gradient (see [23]), they always belong to the same logical time-step. The master CPU takes care of combining partial results.

The actual implementation of the microsimulation was done by defining descendent C++ classes of the C++ base classes provided in the Parallel Toolbox. The underlying communication library has interfaces for both PVM and MPI. A description of the toolbox is beyond the scope of this paper. More information can be found in [12].

5.3 Off-line Load Balancing

We implemented a simple external feedback for the initial static load balancing. During run time we collect the execution time of each link and each intersection (node). The statistics are output to file every 1000 time-steps. For the next iteration run the file is fed back to the initial load balancing algorithm. In this iteration, instead of using the link lengths as load estimate, the actual execution times are used as distribution criterion. Fig. 10 shows the new tiles after such a feedback (compare to Fig. 6), and Fig. 11 shows the improved CPU usage in an example for 4 CPUs.

To verify the impact of this approach we monitored the execution times per time-step throughout the simulation period. Figure 12 depicts the results of one of the iteration runs. For iteration step 1, the load balancer uses the link lengths as criterion. The execution times are low until the first grid-locks appear around 7:30 am. A grid-lock is a traffic situation in which vehicles get stuck in dense traffic jams which cannot be resolved anymore. The execution time increased fivefold from 0.04 sec to 0.2 sec. In iteration 2 the execution time is almost independent of the simulation time. Note that due to the equilibration, the execution time for early simulation hours increased from 0.04 sec to 0.06 sec, but this effect is more than compensated later on.

The figure also contains plots for later iterations (11, 15, 20, and 40). The improvement of execution times is mainly due to the route adaptation process: all grid-locks have disappeared and the average vehicle density is much lower.

6 "Dallas" Performance

For the Dallas study, the trip file contained 300,000 trips occupying approximately 250 MByte of disk space (120 MByte in a slightly compressed binary format, i.e. after using gzip). The route planner network comprised 9864 nodes and unidirectional 24662 links. The simulation network itself consisted of 2276 nodes and 6124 unidirectional links covering an area of about five miles × five miles in the center of the Dallas street network. The original routes generated for the planning network were truncated to the simulation network by a pre-processor. A typical iteration step takes about 6-8 minutes for pre-processing,

30-35 minutes for simulation using eight CPUs (250 MHz) on SUN Enterprise 4000, and 15-20 minutes for re-planning on one CPU. Remember that we need of the order of 50 runs through this; in consequence, we need about 2 days of continuous computing on our 8 CPU machine for one such iteration series.

7 Performance prediction for Portland (Oregon)

For the Dallas case, we have not just done the above iterations, but we have used the set-up for extensive studies of the computational performance. This included a systematic derivation of an equation for the computational performance, including such elements as CPU time, communication time (start-up and bandwidth), competition for bandwidth, etc. This investigation, with the results of the above graph partitioning investigations, was used to make estimates for our Portland problem. The result, for the 200 000 link network, can be seen in Fig. 13. The figure shows predictions for the so-called real time ratio, which says how much faster than reality the simulation is. This is for example interesting for real time applications, since one does not want the forecast to be slower than reality. The thick lines in the figure refer to a 250 MHz SUN Enterprise 4000 (black) and a hypothetical machine consisting of the same CPUs but a two-dimensional communciations topology. The Enterprise 4000 has a fast backplane, but it is still a bus communications system, thus levelling out at about 50 CPUs without getting faster than about twice as fast as real time. The 2-d communcations topology does not have this problem, and speed-up is nearly linear. (Note, however, that the Enterprise 4000 does not accept more than 14 CPUs; Enterprise computers which accept more CPUs also have faster backplanes.)

The prediction means the following: Let us assume we want to look at 24 hours of traffic, after 50 iterations. Using one CPU, it would take 500 days to get the desired result. Using 500 CPUs, it would still take a day.

Fig. 14 shows preliminary actual real time ratio measurements for the predicted situation, i.e. the micro-simulation running on the 200 000 links Portland network. The Origin 2000 is a supercomputer with a more powerful communications technlology than our Enterprise 4000, the predicted near-linear speed-up is confirmed, and computational speeds may even be faster than predicted due to additional optimizations in the code. "Preliminary" in the above sentence means that no vehicles were in the simulation. In our experience, the addition of actual vehicles does not slow down the code enourmously.[2]

8 Summary and Outlook

We presented a framework for dynamic traffic assignment. The fast implementation of the traffic-simulation and the use of parallel computers allowed us to compute a self-consistent route-set within two days of computing on an 8 CPU machine for a problem with about 6 000 links. Moving to a realistic representation of a medium-sized complete city such as Portland/Oregon (1.5 mio people), one would need a 250 CPU machine to obtain the same result after a similar computing time. The result is based on geometric domain decomposition of the geographical area, together with an adaptive load balancing and knowledge about the efficiency of the domain decomposition.

[2] We will not be able to measure the scenario with vehicles since our computing access has been revoked.

9 Acknowledgements

We thank A. Bachem, R. Schrader and C. Barrett for supporting MR's work as part of the traffic simulation efforts in Cologne ("Verkehrsverbund Verkehrsimulation und Umweltwirkungen NRW") and Los Alamos (TRANSIMS). Computing time on the SGI-1 Challenger of the Regionales Rechenzentrum Köln and on the workstation cluster at TSA-DO/SA is gratefully acknowledged. The work of MR was supported in part by the "Graduiertenkolleg Scientific Computing Köln". Part of this work was performed at Los Alamos National Laboratory, which is operated by the University of California for the U.S. Department of Energy under contract W-7405-ENG-36 (LA-UR 98-2250).

References

[1] TRANSIMS, TRansportation ANalysis and SIMulation System, Los Alamos National Laboratory, Los Alamos, U.S.A. See http://transims.tsasa.lanl.gov.

[2] Y. Sheffi. *Urban transportation networks: Equilibrium analysis with mathematical programming methods.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1985.

[3] DYNAMIT, see http://its.mit.edu.

[4] R.H.M. Emmerink, K.W. Axhausen, P. Nijkamp, and P. Rietveld. Effects of information in road transport networks with recurrent congestion. *Transportation*, 22:21, 1995.

[5] H.S. Mahmassani, T. Hu, and R. Jayakrishnan. Dynamic traffic assignment and simulation for advanced network informatics (DYNASMART). In N.H. Gartner and G. Improta, editors, *Urban traffic networks: Dynamic flow modeling and control.* Springer, Berlin/New York, 1995.

[6] K. Nagel. Individual adaption in a path-based simulation of the freeway network of Northrhine-Westfalia. *International Journal of Modern Physics C*, 7(6):883, 1996.

[7] C. Gawron. An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model. In Wolf and Schreckenberg [25], pages 469–474.

[8] H. A. Rakha and M. W. Van Aerde. Comparison of simulation modules of TRANSYT and INTEGRATION models. In *Traffic Flow Theory and Traffic Flow Simulation Models*, volume 1566 of *Transportation Research Record*, pages 1–7. Transportation Research Board, Washington, D.C., 1996.

[9] R.J. Beckman et al. TRANSIMS–Release 1.0 – The Dallas-Fort Worth case study. Los Alamos Unclassified Report (LA-UR) 97-4502, Los Alamos National Laboratory, see http://transims.tsasa.lanl.gov, 1997.

[10] J. Esser and K. Nagel. Census-based travel demand generation for transportation simulations. In M. Schreckenberg al, editor, *Proceedings of the workshop "Traffic and Mobility*, Aachen, Germany, Sep/Oct 1998. Also Los Alamos Unclassified Report LA-UR 99-10, see http://www.santafe.edu/~kai/papers.

[11] K. Nagel. Experiences with iterated traffic microsimulations in Dallas. In Wolf and Schreckenberg [25], pages 199–214.

[12] M. Rickert. *Traffic simulation on distributed memory computers.* PhD thesis, University of Cologne, Cologne, Germany, 1998. See http://www.zpr.uni-koeln.de/~mr/dissertation.

[13] R. R. Jacob, M. V. Marathe, and K. Nagel. A computational study of routing algorithms for realistic transportation networks. *ACM Journal of Experimental Algorithms*, in press. Also Los Alamos Unclassified Report LA-UR 98-2249, see http://www.santafe.edu/~kai/papers.

[14] K. Nagel. Particle hopping models and traffic flow theory. *Physical Review E*, 53(5):4655, 1996.

[15] K. Nagel. From particle hopping models to traffic flow theory. *Transportation Research Records*, 1644, in press.

[16] S. Krauß. *Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics.* PhD thesis, University of Cologne, Cologne, Germany, 1997.

[17] M. Rickert, K. Nagel, M. Schreckenberg, and A. Latour. Two lane traffic simulations using cellular automata. *Physica A*, 231:534, 1996.

[18] K. Nagel, P. Stretz, M. Pieck, S. Leckey, R. Donnelly, and C. L. Barrett. TRANSIMS traffic flow characteristics. Los Alamos Unclassified Report (LA-UR) 97-3530, Los Alamos National Laboratory, see http://www.santafe.edu/~kai/papers, 1997. Earlier version: Transportation Research Board (TRB) preprint 981332.

[19] M. Rickert and P. Wagner. Parallel real-time implementation of large-scale, route-plan-driven traffic simulation. *International Journal of Modern Physics C*, 7(2):133–153, 1996.

[20] C. Gawron, M. Rickert, and P. Wagner. Real-time simulation of the german autobahn network. In F. Hoßfeld, E. Maehle, and E.W. Mayer, editors, *Proc. of the 4th Workshop on Parallel Systems and Algorithms (PASA '96)*. World Scientific Publishing Co., 1997.

[21] K Nagel, M Rickert, and C L Barrett. Large-scale traffic simulations. In J. M. L. M. Palma and J. Dongarra, editors, *Vector and Parallel Processing – VECPAR'96*, volume 1215 of *Lecture Notes in Computer Science*, pages 380–402. Springer, 1997.

[22] Metis library. http://www-users.cs.umn.edu/~karypis/metis/metis.html.

[23] K. Nagel and A. Schleicher. Microscopic traffic modeling on parallel high performance computers. *Parallel Computing*, 20:125–146, 1994.

[24] K. Nagel, M. Rickert, R. Frye, P. Stretz, P. M. Simon, R. Jacob, and C. L. Barrett. Regional transportation simulations. In *Proceedings of the Advanced Simulation Technologies Conference*, Boston, MA, U.S.A., 1998. The Society for Computer Simulation International.

[25] D.E. Wolf and M. Schreckenberg, editors. *Traffic and granular flow '97.* Springer, Heidelberg, 1998.

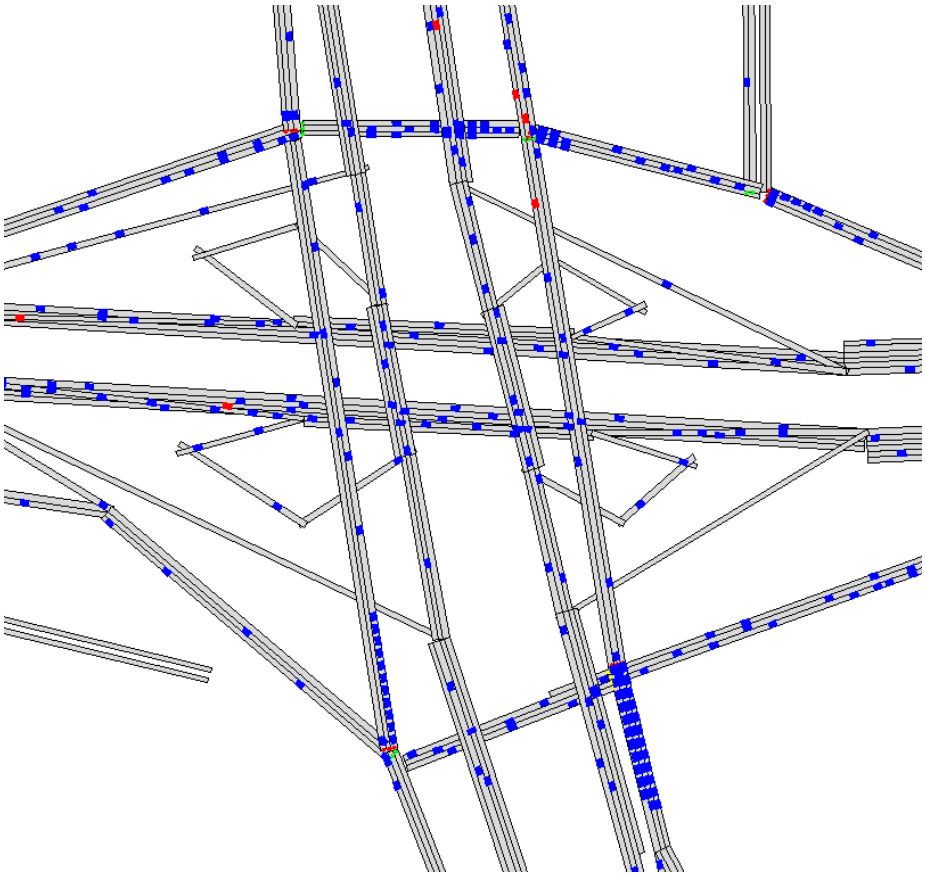

Figure 1: Virtual reality transportation system representation.

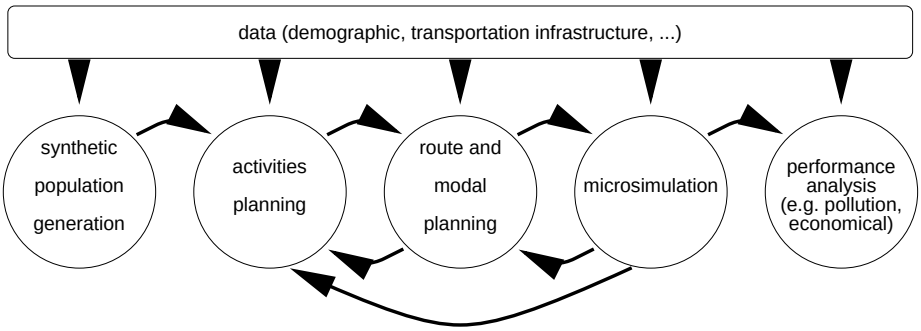

Figure 2: TRANSIMS design

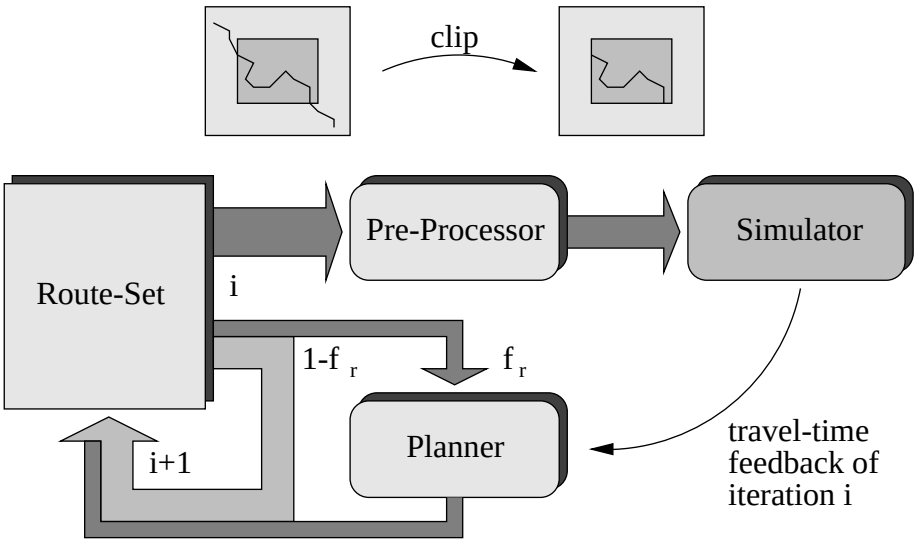

Figure 3: Iterative assignment with simulation feedback. For iteration $i+1$ the fraction f_r of the previous route-set is re-planned by the router using link travel-times of iteration i. The remaining fraction $1-f_r$ is simply reused. Before the route-set is fed into the simulation it is clipped to the boundaries of the study-area.

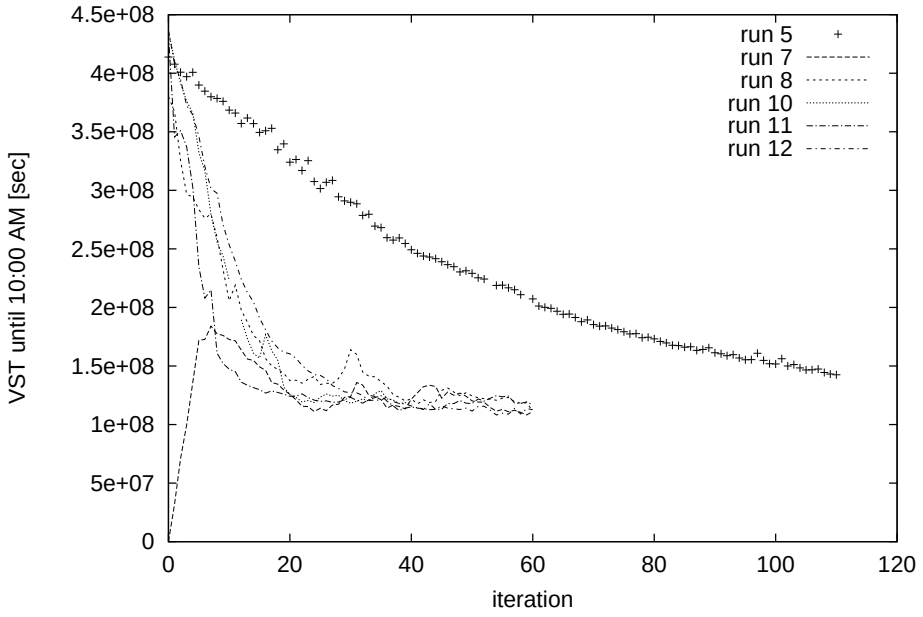

Figure 4: Relaxation of the accumulated travel-time.

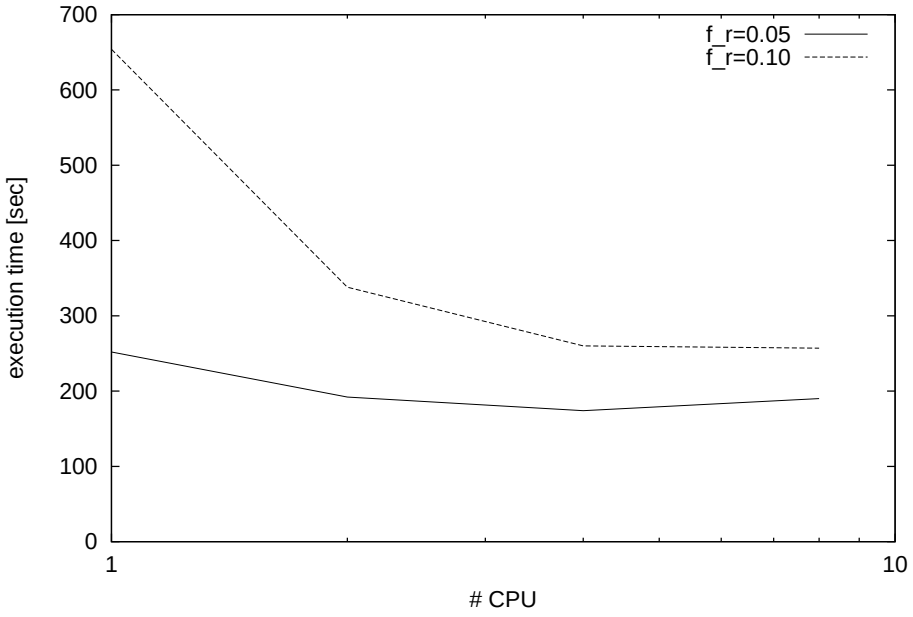

Figure 5: Execution times of the Route Planner

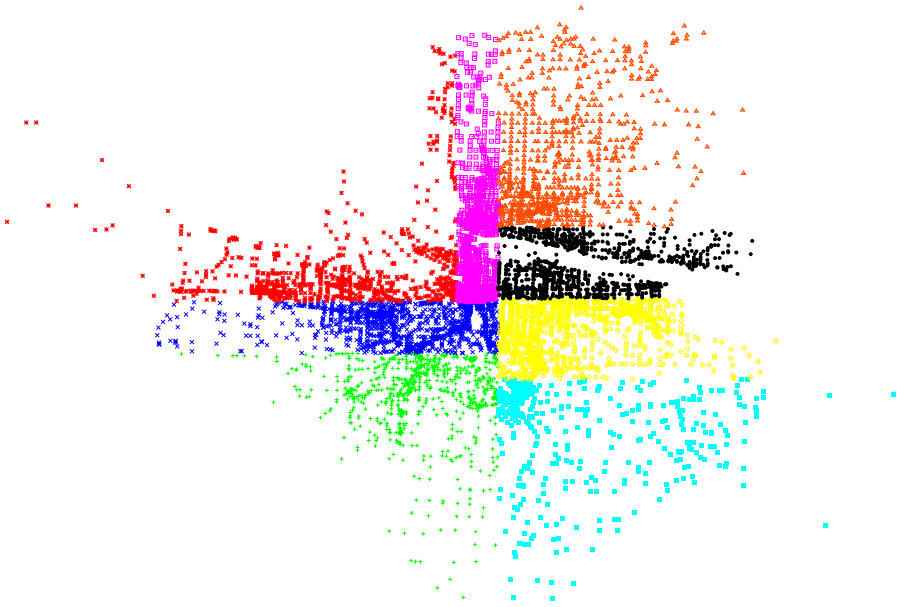

Figure 6: Orthogonal bi-section

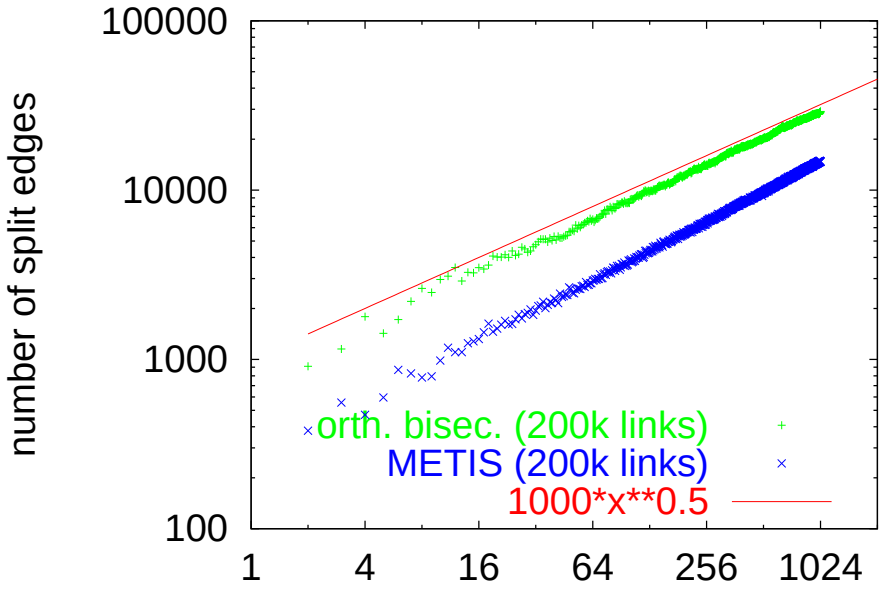

Figure 7: Number of split edges

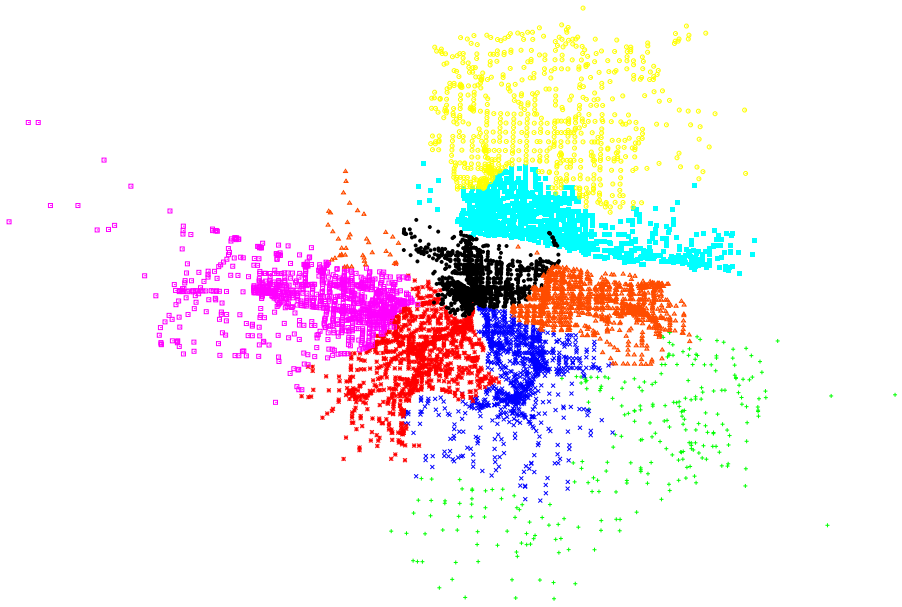

Figure 8: Partitioning by METIS

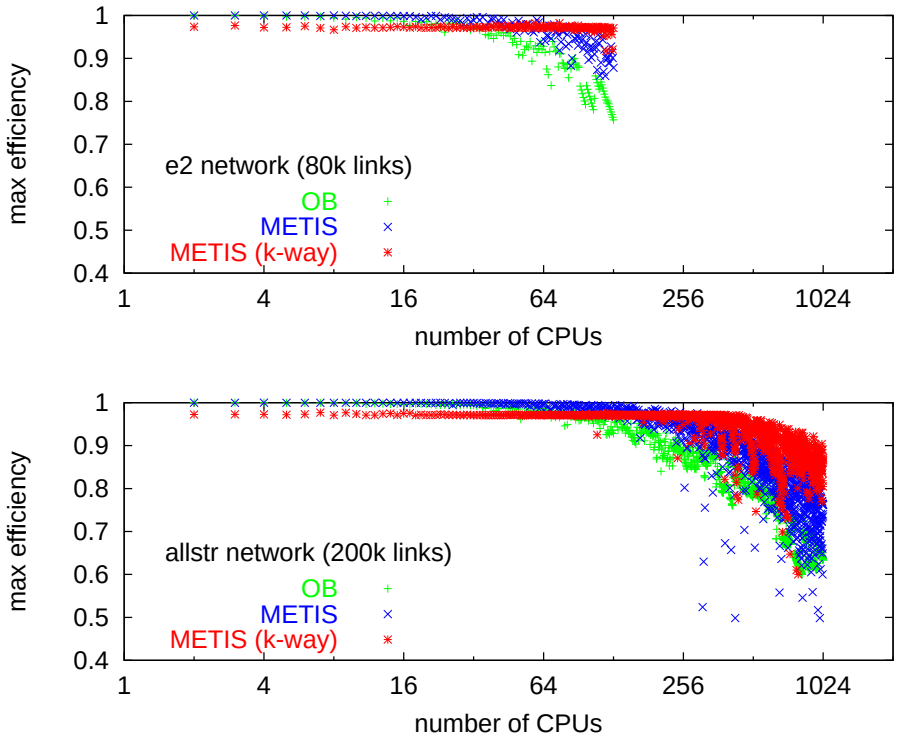

Figure 9: *Top:* Theoretical efficiency for Portland network with 20 000 edges. *Bottom:* Theoretical efficiency for Portland network with 200 000 edges.

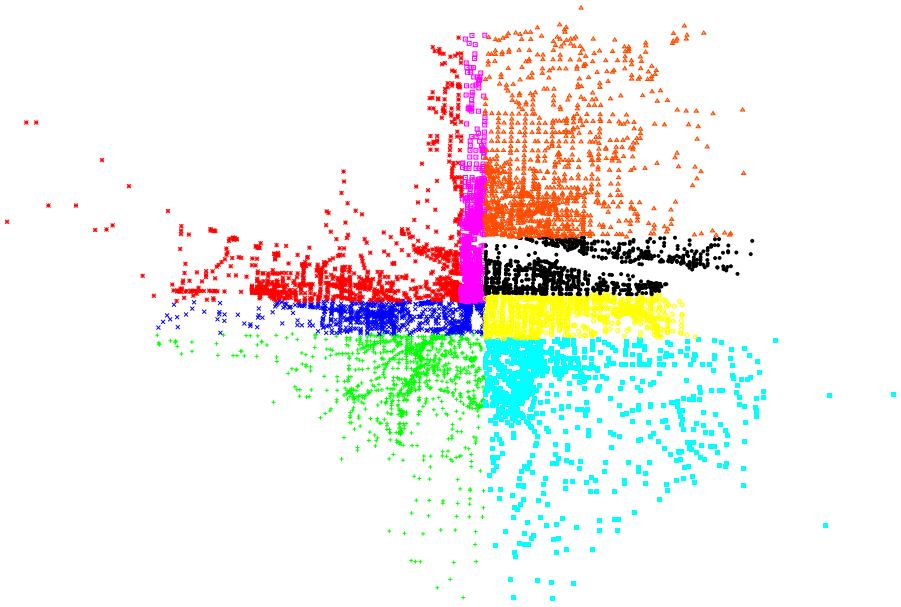

Figure 10: Partitioning after load balancing. Compare to Fig. 6.

Figure 11: *Top:* CPU usage without loadbalancing feedback. *Bottom:* CPU usage with loadbalancing feedback. Note that the idle time, summed over all CPU, is much shorter.

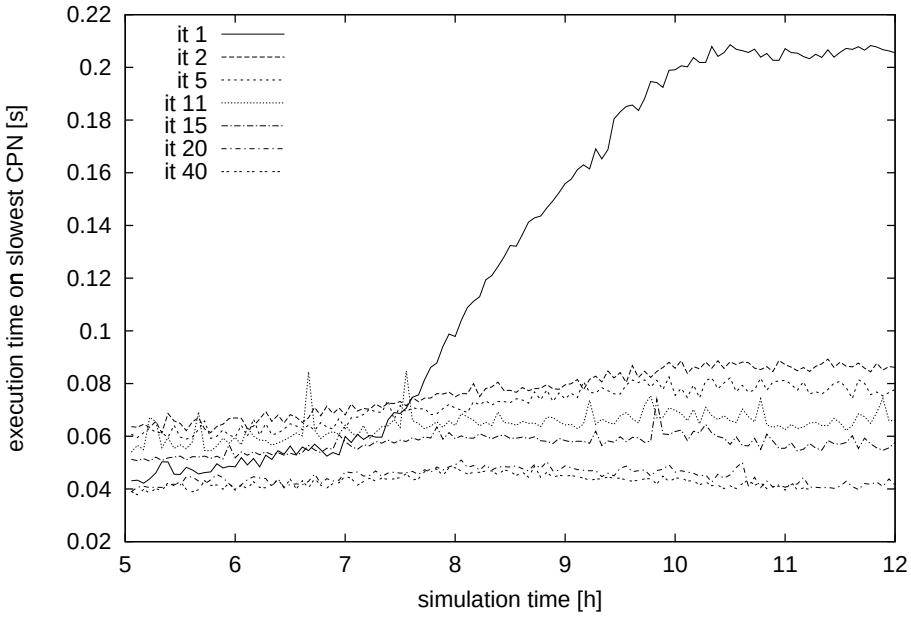

Figure 12: Execution times with external load feedback

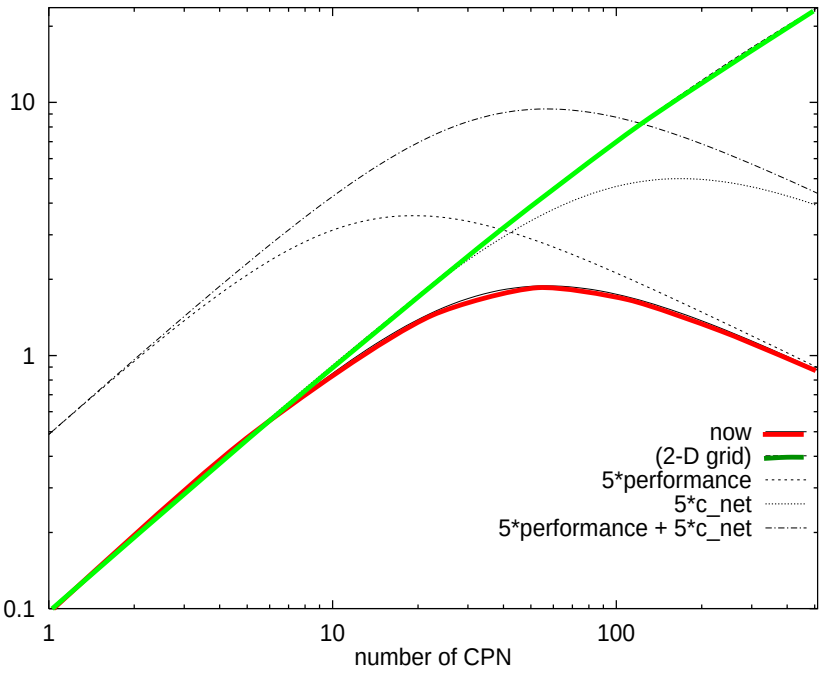

Figure 13: Performance predictions. From [24].

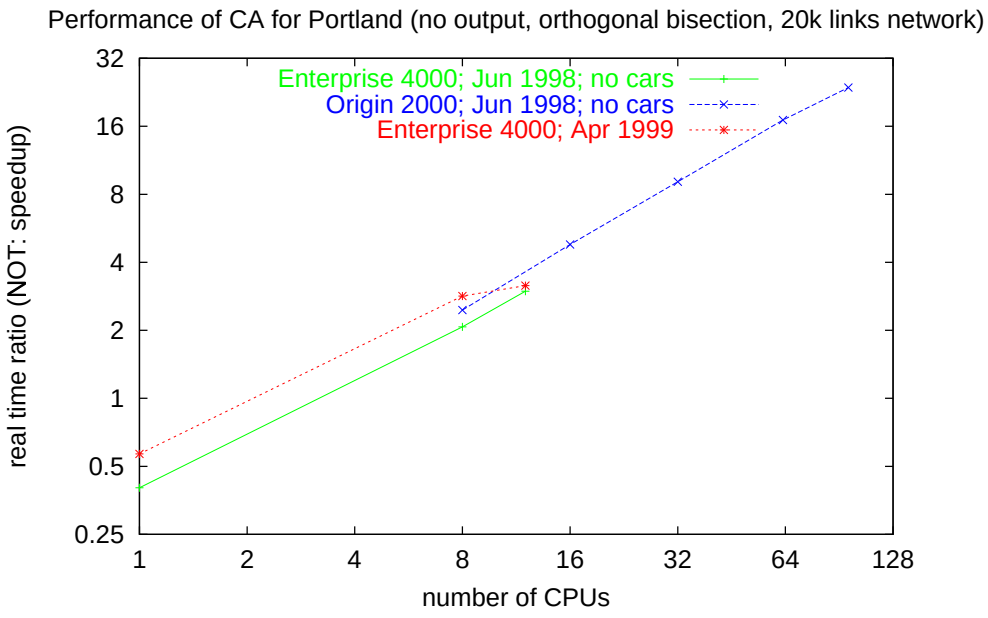

Figure 14: Performance, measurements on Sparc Enterprise 4000, and on Origin 2000.