Determination of traffic control tables by HPC

Eligius M.T. Hendrix, Siham Tabik[1] and Rene Haijema[2]

Resumen— **The concept of traffic control tables (TCT) for an intersection is sketched and a Stochastic Dynamic Programming model is outlined. The determination of a TCT by dynamic programming becomes more cumbersome if more traffic flows and combination of lights are taken into account. This paper explains how High Performance Computing (HPC) can be essential to do this job and sketches the challenges of this research question.**

Palabras clave— **Stochastic Dynamic Programming, traffic control, parallel implementation, Markov chain**

I. Introducción

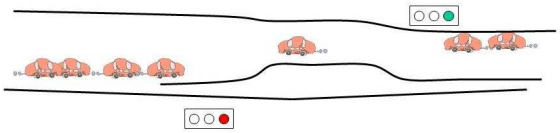

Fig. 1. Situation F2C2; at most one of two flows gets priority

Traffic lights are introduced at the beginning of the previous century to make road traffic safer, at places where traffic from different directions cross the same road segment, called the intersection or crossing. By giving right of way to traffic in some direction(s), cars approaching from other directions need to wait before they get priority. By controlling the traffic lights, the overall delay or waiting time of the cars can be kept to a minimum. In literature, the problem is studied by queueing theorist as well as engineers (see [5], [6], [9],[8], [10]). An optimal dynamic policy is not reported except in [4].

The basis of optimal traffic control on a single intersection is what we call a traffic control table (TCT) that prescribes which combination of flows should be given the right of way given the amount of cars waiting in every queue. For illustration consider the simple situation in Figure 1, called in [3] the F2C2 case for having 2-Flows in 2-Combinations. Either the left, or right flow has a green light, or all lights are red to clear the intersection. For the ease of reasoning, we abstract here from using amber lights.

Table I illustrates the concept of a TCT. It marks the decision of which light to set on green, given the queue length of both queues, when the light is in the all-red state. In this example $\lambda_2 > \lambda_1$, such that on equal queue length, it is convenient to give flow 2 the right of way. [4] model the generation of such a table as a Stochastic Dynamic Programming (SDP) problem in order to find that TCT that minimizes the expected total waiting time in the system.

TABLA I
TCT for F2C2 when lights show all-red

q_1 q_2	0	1	2	3	4	5
0	2	2	2	2	2	2
1	1	2	2	2	2	2
2	1	1	2	2	2	2
3	1	1	1	2	2	2
4	1	1	1	1	2	2
5	1	1	1	1	1	2

In Section II, the SDP model is outlined and the iterative process to obtain traffic control tables from that. Section III describes the parallel programming approach used to exploit multicore systems. Section IV provides the experimental set up where a computational illustration is given in Section V. Finally, Section VI concludes.

II. TCT by Stochastic Dynamic Programming

There are numerous ways to model traffic flows. We focus here on a Markov chain view with time slots are thought of as to be that big that one car can pass by on a green light, usually taken as two seconds. The state is described by the vector $(\mathbf{q}, l)$, where the vector $\mathbf{q}$ tracks the number of cars in each queue and $l \in \{0, \ldots, ncomb\}$ indicates the state of the light, i.e. which of the $ncomb$ combination has right of way ($l = 0$ represents the all-red state).

The probabilities of going from one state to the other depend on the TCT as well as on the probabilities λ_j of a car arriving at the queues j (for all $j \in \{1, \ldots, nflow\}$. In order to get a finite state space to allow numerical computations the queue length is truncated to a maximum size Q, such that $q_j \in \{0, \ldots, Q\}$.

For the F2C2 case of Figure 1, this means that the state space is 3-dimensional: (q_1, q_2, l). The number of possible states is $ns = (ncomb{+}1){\times}(Q{+}1)^{nflow} = 3(Q+1)^{nflow}$. Consider the F4C2 case of Figure 2. The state space is 5 dimensional and the number of states is $ns = 3(Q+1)^4$. We observe that the number of states grows exponentially fast in the number of queues; this called the curse of dimensionality in solving an SDP problem. Nevertheless up to the F4C2 case, Haijema concludes that an optimal policy can relatively easily be computed on a PC with a single processor. Although the curse of dimensionality is not resolved by HPC, HPC may stretch the computational limit beyond the F4C2 case.

To find a TCT that minimizes the expected wait-

[1]Dpto. Arquitectura de Computadores, Univ. De Málaga, e-mail: eligius, stabik @uma.es.
[2]Operations Research and Logistics group, Wageningen University, e-mail: rene.haijema@wur.nl.

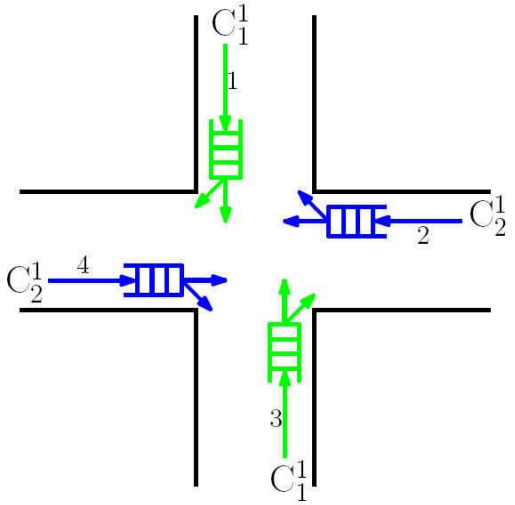

Fig. 2. Situation with 4 flows and 2 symmetric combinations

ing time we apply the so-called value iteration (VA) algorithm (for details see [7] and [3]). First a cost function $C(\mathbf{q}, l) = \sum_{j=1}^{nflow} q_j$ is defined that captures the waiting time during the coming time slot for the current state $(\mathbf{q}, l)$. Note that by Little's law minimizing the expected number of cars waiting also minimizes the overall expected waiting time. Next a so-called value vector $V_n(\mathbf{q}, l)$ that gives a valuation for each state over the next n time slots, is to be computed for $n = 1, 2, 3, \ldots$. Clearly the cost over 0 periods is zero, hence we start the value iteration algorithm with $n = 0$ and $V_n(\mathbf{q}, l) = 0$ for all states $(\mathbf{q}, l)$. Then one determines iteratively optimum decisions a such that

$$V_{n+1}(\mathbf{q}, l) = C(\mathbf{q}, l) + \min_a E_{e|a} V_n(T(\mathbf{q}, l, a, e))$$

where T is a transformation function that gives the state at which to arrive when at the current state, the decision a is taken and arrival event e happens. The whole is sketched in Figure 3. Notice that the number of possible events is 2^{nflow}, as at each queue, a car may arrive or not. The probabilities are determined from the vector of traffic intensities $(\lambda_1, \ldots, \lambda_{nflow})$. If $l = 0$, i.e. all lights are red, the decision $a \in \{1, \ldots, ncomb\}$, i.e. one of the combinations can be given a green light. In the other cases, there are 2 possibilities; either the light stays as it is, or is put in the all-red state to clear the intersection.

The converging part in the process is the difference $V_{n+1} - V_n$ converging to a constant vector which represents the average waiting time in the system. Practical implementations require the translation of the state $(\mathbf{q}, l)$ to a state number i and vice versa, such that one works with two arrays with elements $V_{n+1}(i)$ and $V_n(i)$. The convergence is measured by keeping hold of the so-called span defined as $span = \max_i(V_{n+1}(i) - V_n(i)) - min_i(V_{n+1}(i) - V_n(i))$. This is illustrated in Figure 4.

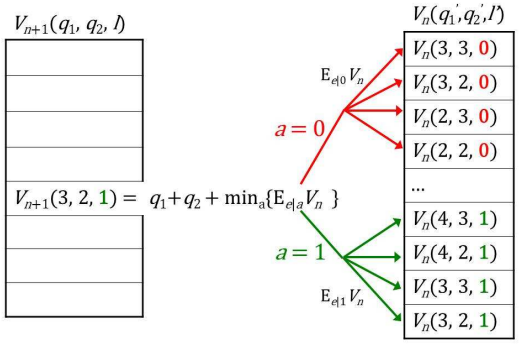

Fig. 3. Value function determination

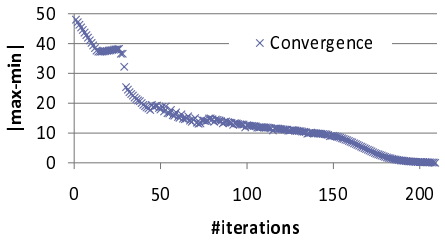

Fig. 4. Convergence of the span for the F4C2 instance, calculated by the parallel code on a quad-socket eight-core Intel X7550 (Beckton).

III. Coding Value iteration

The value iteration process requires running the iterations up to convergence. At each iteration all values for the ns states of V_{n+1} have to be determined. If $l = 0$, this requires looking up $ncomb \times 2^{nflow}$ values in V_n. As we have seen, this is less if one of the combinations is green. We should look up $2^{nflow+1}$ and take the minimum over the two decisions.

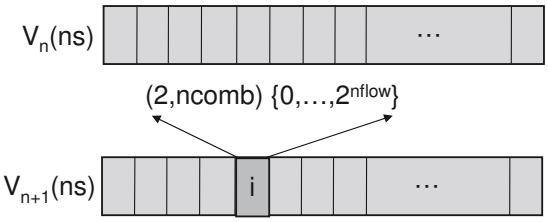

Fig. 5. Each $V_{n+1}(i)$ is determined using 2 or $ncomb \times Q^{nflow}$ elements of V_n, depending on the state of light.

In summary, the iterative process of value iteration for the TCT generation can be sketched as follows:

```
for(i=0;i < nflows;i++)q[i]=1;
while(1){
    for(i=0;i<num_states; i++)
        V_{n+1}[i]=V_n[i];
    for(i=0;i<num_states; i++){
        compute_decision(&light);
        if(light==all_red)
                compute_new_state(ncomb,V_n,V_{n+1})
        else if( light == green at one flow)
                compute_new_state(light,V_n,V_{n+1});
        compute_max&min(V_n);
    }
```

if (max-min<epsilon) **break**;
}

To be able to model scenarios that involve more complex and larger intersections, parallel computing turns out to be essential. For this reason, we developed a first parallel implementation of the TCT computation using a hybrid approach that mixes data-sharing and data-privatizing programming approaches. The resulting parallel implementation allows varying/adjusting the sharing level from 100% sharing, where all threads share all the data to 0% sharing, where each process works on its own data. A preliminary performance analysis of the 100% sharing implementation has shown that the workload per iteration is irregular since cumputing one element in the current value function vector may need from 2×2^{nflow} to $ncomb\times 2^{nflow}$ of elements from the former one. After analyzing many scheduling strategies in combination with different chunk sizes, we found out that by mapping iterations, of a chunk size equal to 1, in a round robin fashion among the participating threads provides the best performance results. In the experimental results section, we only report the scalability obtained by using the aforementioned scheduling methodology. The optimal mapping and chunk size may change when modeling larger number of states.

IV. Experimental Testbed

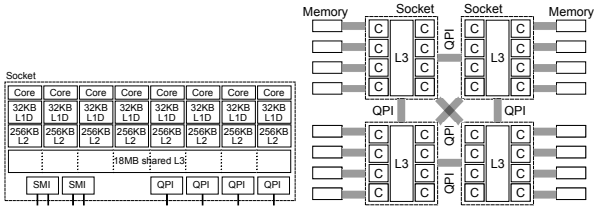

Fig. 6. Block diagrams of (a) single Beckton and (b) quad-socket eigth-core Intel X7550 (Beckton).

All the experiments shown in this paper were conducted on a quad-socket eigth-core Intel X7550 (Beckton). An architectural overview and characteristics of this system appear in Figure 6 and Table II.

A. Intel Quad-socket Eight-core Beckton

The Intel Xeon 6500/7500-series (Beckton) is a Nehalem-based processor with up to eigth cores and uses buffering to support up to 16 DDR3 DIMMS per socket (no FSB). The specific machine we use in the experiments is X7550 that runs at 2.0 GHz, includes eigth cores and is capable to perform around 70 GFLOPS double-precision in the LINPACK benchmark.

Each processor core includes a private 32KB L1 data cache, a private 256KB L2 cache and a shared multi-banked 18MB L3 cache. The processor also has four QuickPath interfaces (QPI), with a speed of 6.4 GT/s, that can be used to interconnect up to eight sockets. The processor cores support a 2-way hyper-threading technology (there are two hardware thread contexts per core).

The system used in the experiments includes four sockets containing an X7550 processor each one. The sockets are fully interconnected using QPI links and share a total of 128GB main memory. Fig. 6 (a) shows a block diagram of the X7550 processor, and Fig. 6 (b) shows a block diagram of the complete system.

V. Performance Results

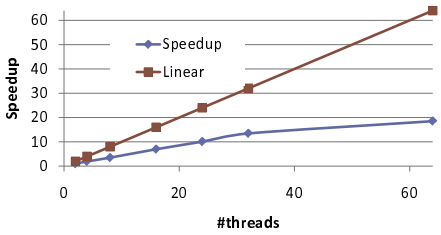

Fig. 7. Speedup of the data-sharing parallel implementation that exploits STM capabilities on quad-socket eigth-core Intel X7550 (Beckton).

In this section we discuss only the performance of the data-sharing part of the hybrid implementation since we are currently working on optimizing the data-privatizing part which is based on message passing. Performance evaluations were carried out using the following configuration. For thread handling and synchronization, we used OpenMP pragmas and directives included in icc compiler version 10.1 [2], [1]. We also used the maximum optimization compilation option -O3. To exploit SMT support and increase the positive effect of data-sharing in the considered multi-socket, multi-core platform, we run two threads per core and bind them on these cores along the execution. For the evaluation process, a F4C2 instance was used with $Q = 5$, $nflow = 4$, number of combination=2, $epsilon = 0.1$ and $\lambda = (0.2, 0.2, 0.2, 0.2)$. Notice that this instance implies having 85683 states.

In general, the parallel implementation using data-sharing shows very good scalability up to one Socket, reaching speedups equal to $2 \times \#cores$ on 8 cores, as shown Figure 7. However, when the evaluation includes cores of different Sockets, the speedup starts to decay substantially. This is mainly due to the increase of L3 misses and wait times.

VI. Conclusions

The process of value iteration can be used to derive Traffic Control Tables for intersections that minimize expected waiting time. Due to higher traffic intensity, and more complex situations, the number of necessary states to be considered in each iteration grows polynomially in the queue length and exponentially in the number of flows. Use of multicore systems can be a promising approach and is essential to provide benchmarks for heuristic approaches. A hybrid parallel implementation has been developed to

TABLA II

ARCHITECURAL SUMMARY OF THE MULTI-CORE PLATFORM USED IN THE EXPERIMENTS

Intel X7550	4 sockets				
cores/socket	**SMT**	**L1D cache**	**L2 cache**	**L3 cache**	**memory**
8	yes	8×32KB	8×256KB	1×18MB	128GB

exploit the potential of multi-socket multi-core architectures. Preliminary speedups of the data-sharing implementation have been measured for a medium sized instance called F4C2.

Currently, we are exploring new techniques such as software prefetching and cache blocking to further improve the performance of the data-sharing implementation at one Socket level. In addition, we are working on optimizing the communications in the data-privatizing implementation to improve the performance among multiple Sockets. The intention is to apply the techniques to the F12C4 case given in Figure 8. for which no optimum TCT has been derived yet. Updating one value of the value function requires the evaluation of several times 2^{12} states due to the possible events. Moreover, using a minimum queue length of $Q = 2$, this has to be done for $5 \times 3^{12} = 2.7$ mln states. Future investigation will look at using interpolation to keep the number of states limited to that number.

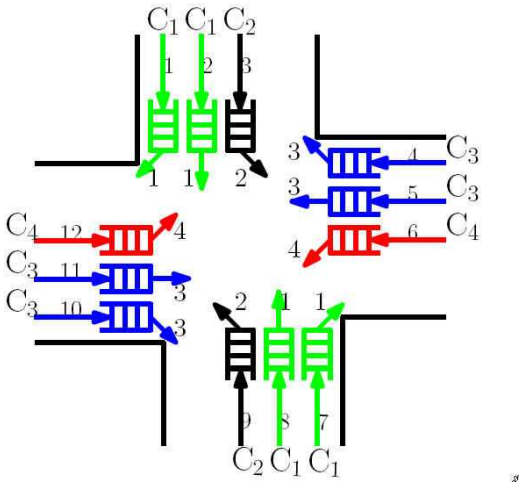

Fig. 8. Instance of intersection with 12 flows, 4 combinations

AGRADECIMIENTOS

This work is supported by grants from the Spanish Ministry of Science and Innovation (TIN2008-01117, TIN2006-01078), Junta de Andalucía (P08-TIC-3518), in part financed by the European Regional Development Fund (ERDF). Eligius Hendrix is fellow of the Spanish "Ramon y Cajal" contract program and Siham Tabik of the "Juan de la Cierva" program, co-financed by the European Social Fund.

REFERENCIAS

[1] *The OpenMP API specification for parallel programming*, http://openmp.org/wp/openmp-specifications/.

[2] *Intel Compilers for Linux*, http://software.intel.com/en-us/articles/intel-c-compiler-professional-edition-for-linux-documentation/ (2009).

[3] R. Haijema, *Solving large structured markov decision problems for perishable inventory management and traffic control*, Ph.D. thesis, Univeristy of Amsterdam - Tinbergen Institute - Amsterdam School of Economics, 12 2008.

[4] R. Haijema and J. van der Wal, *An MDP decomposition approach for traffic control at isolated signalized intersections*, Probability in the Engineering and Informational Sciences **22** (2008), no. 4, 587–602.

[5] G. F. Newell, *Approximation methods for queues with applications to the fixed-cycle traffic light*, SIAM Review **7** (1965), no. 2, 223–240.

[6] M. Papageorgiou, C. Diakaki, V. Dinopoulou, A. Kotsialos, and Y. Wang, *Review of road traffic control strategies*, Proc. of the IEEE, vol. 91, IEEE, 2003, pp. 2043–2067.

[7] M. L. Puterman, *Markov decision processes: Discrete stochastic dynamic programming*, Wiley Series in Probability and Mathematical Statistics, 1994.

[8] M. S. van den Broek, J. S. H. van Leeuwaarden, I. J. B. F. Adan, and O. J. Boxma, *Bounds and approximations for the fixed-cycle traffic-light queue*, Transportation Science **40** (2006), 484–496.

[9] J. S. H. van Leeuwaarden, *Delay analysis for the fixed cycle traffic light queue*, Transportation Science **40** (2006), no. 2, 189–199.

[10] M. Wiering, J. van Veenen, J. Vreeken, and A. Koopman, *Intelligent traffic light control*, technical report UU-CS-2004-029, Institute of information and computing sciences, Utrecht University, 2004.