Speeding Up the Evaluation of a Mathematical Model for VANETs Using OpenMP

Carolina García-Costa, Juan Bautista Tomás-Gabarrón,
Esteban Egea-López, and Joan García-Haro

Department of Information and Communications Technologies,
Universidad Politécnica de Cartagena (UPCT),
Plaza del Hospital 1, Cartagena, Spain
{carolina.garcia,juanba.tomas,esteban.egea,joang.haro}@upct.es

Abstract. Vehicular Ad-hoc Networks (VANETs) are having a significant impact on Intelligent Transportation Systems, specially on the improvement of road safety. Cooperative/Chain Collision Avoidance (CCA) application comes up as a solution for decreasing accidents on the road, therefore it is highly convenient to study how the system of vehicles in a platoon will behave at different stages of technology deployment until full penetration in the market. In the present paper we describe an analytical model to compute the average number of accidents in a chain of vehicles. The use of this model when the CCA technology penetration rate is not 100% leads to a vast increase in the number of operations. Using the OpenMP directives for parallel processing with shared memory we achieve a significant reduction in the computation time consumed by our analytical model.

Keywords: OpenMP, VANET, supercomputing, Cooperative/Chain Collision Avoidance application.

1 Introduction

Vehicular networks, also known as VANETs, are defined as *ad-hoc* mobile networks with two main communication features. On the one hand, VANETs are in charge of transmitting information among vehicles (V2V communications). In this first case, cars carry out the information interchange without any infrastructure support for regulating the access. On the other hand, an intercommunication among vehicles and infrastructures also exists (V2I communications), making possible a connection through cars and a backbone network, reaching in this way those vehicular entities allocated out of the direct communication range.

One of the aims of vehicular networks development is the improvement of road safety. The main goal of these innovative systems is to provide drivers a better knowledge about road conditions, decreasing the number of accidents and their severity, and simultaneously aiding to a more comfortable and fluent driving. Other vehicular applications are also considered, such as Internet access, driving cooperation and public information services support.

A Cooperative/Chain Collision Avoidance (CCA) application [1] uses VANET communications for warning drivers and decreasing the number of traffic accidents. CCA

takes advantage of vehicles with cooperative communication skills, in a way that these cars are able to react to possible accident risks or emergence situations. The CCA mechanism generates an encapsulated notification which is sent as a message through a one-hop communication scheme to all vehicles within a potential danger coverage (relay schemes are also possible). It should be noted that the establishment of this VANET application will be deployed gradually, equipping vehicles with the proper hardware and software so as they can communicate in an effective way within the vehicular environment.

In our research we consider a platoon (or chain) of N vehicles following a leading one. The leading vehicle stops instantly and the following vehicles start to brake when they are aware of the risk of collision, because of a warning message reception or the perception of a reduction in the speed of the vehicle immediately ahead. To test the worst case situation, vehicles cannot change lane or perform evasive maneuvers.

We have developed a first approach mathematical model to calculate the average percentage of accidents in the platoon, varying the number of considered vehicles, their average speed, the average inter-vehicle spacing and the penetration ratio of the CCA technology. Specifically when the CCA penetration ratio is taken into account, the growth in the number of operations of the analytical model is such that the sequential computation of a numerical solution is no longer feasible. Consequently, we resort to the use of the OpenMP parallelization techniques for solving those computational cases considered as unapproachable by means of sequential procedures.

Additionally, we execute our programs in the Ben-Arabi Supercomputing environment [2], taking the advantage of utilizing the fourth fastest Supercomputer in Spain. In the current work we show how the parallelization techniques coordinated with supercomputing resources make the simulation process a more suitable and efficient one, allowing a thorough evaluation of the CCA application.

The remainder of this paper is organized as follows. In Section 2 we briefly review the related work. In Section 3 the OpenMP environment is briefly reviewed and the Ben-Arabi Supercomputer architecture introduced. A description of the mathematical model, its implementation and parallelization are provided in Sections 4 and 5. Finally, some results are shown and discussed in Section 6 to illustrate the performance of the resulting parallel algorithm. In this section it is also described our unsuccessful experience of using the MPI parallelization technique to further reduce the computation times. Conclusions and future work are remarked in Section 7. Let us mention that this paper is an extension of the work presented by the authors in [3].

2 Related Work

So far, most typical High Performance Computing (HPC) problems focused on those fields related with certain fundamental problems in several areas of science and engineering. Other typical applications are the ones related to commerce, like databases and data mining [4]. That is the reason why we consider our VANET mathematical model approximation as a non-classical issue to be solved under HPC conditions, contributing to extend the use of supercomputing to other fields of interest.

In the implementation of our mathematical model we parallelize a sparse matrix-vector multiplication. This operation is considered as a relevant computational kernel in scientific applications, which performs not optimally on modern processors because of the lack of compromise between memory and computing power and irregular memory access patterns [5]. In general, we find quite a lot of done work in the field of sparse matrix-vector multiplications using parallelization techniques [6], [7], [8]. These papers study in depth the optimal performance of this operation, but in this paper, we show that even using a simpler parallelization routine, the computation time is noticeably shortened.

Several mathematical models have been developed to study different aspects of VANETs. Most of them are related with the vehicle routing optimization [9], [10], the broadcasting methods [11], [12], [13], the mobility of vehicles [14], [15] and the communication delay time [16], [17], [18]. Other related VANET issues have been studied as well, like network connectivity [19], or survivability [20]. In this paper we focus on collision models for a chain of vehicles, particularly those based on physical parameters to assess the collision process itself [21], [22], [23].

However in an attempt of searching related work we find that few work has been done specifically regarding to the parallelization of these VANET mathematical models, strictly speaking. Moreover, to the best of our knowledge, only the vehicle routing problem has been approached using parallelization techniques [24], [25], [26].

Summing up, in this paper we describe a preliminary model (although computationally expensive) for a CCA application to compute the number of chain collisions and we address the benefits of using parallelization techniques in the VANET field.

3 Supporting Tools

3.1 The OpenMP Technique

OpenMP is a well-known open standard for providing parallelization mechanisms to multiprocessors with shared memory [27]. OpenMP API supports shared memory programming, multi-platform techniques for the programming languages like Fortran, C and C++, and for every architecture including Unix and Windows platforms. OpenMP is a scalable and portable model developed for hardware and software distributors which provides shared memory programmers with a simple and flexible interface for developing parallel applications which can run not only in a personal computer but also in a supercomputer.

OpenMP uses the parallel paradigm known as *fork-join* with the generation of multiple threads, where a heavy computational task is divided into k threads (*forks*) with less weight and afterwards it collects their results and combines them at the end of the execution in a single result (*join*).

The master thread runs sequentially till it finds an OpenMP guideline and since this moment a bifurcation is generated with the corresponding slave threads. These threads can be distributed and executed in different processors, decreasing in this way the execution time.

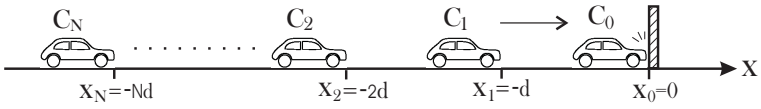

Fig. 1. The scenario under consideration. *d* is the average inter-vehicle distance.

3.2 The Ben-Arabi Supercomputer

Our model is executed under the Ben-Arabi supercomputer resources, which is placed in the Scientific Park of Murcia (Spain). The Ben-Arabi system consists of two different architectures; on the one hand the central node HP Integrity Superdome *SX*2000 with 128 cores of the Intel Itanium-2 dual-core Montvale (1.6 Ghz, 18 MB of cache L3) processor and 1.5 TB of shared memory, called Ben. On the other hand, Arabi is a cluster consisting of 102 nodes, which offers a total of 816 Intel Xeon Quad-Core E5450 (3 GHz y 6 MB of cache L2) processor cores and a total of 1072 GB of shared memory.

We run our mathematical model within a node of the Arabi cluster environment using 2, 4 and 8 processors in order to compare the resulting execution times. Les us remark that we are using a shared memory parallelization technique, so we are not allowed to combine the use of processors from different nodes.

Next we summarize the technical features of the cluster:

- Capacity: 9.72 Tflops.
- Processor: Intel Xeon Quad-Core E5450.
- Nodes number: 102.
- Processors number: 816.
- Processors/Node: 8.
- Memory/Node: 32 nodes of 16 GB and 70 of 8 GB.
- Memory/Core: 3 MB (6 MB shared among 2 cores).
- Clock frequency: 3 Ghz.

4 Model Description

We are interested in evaluating the performance of a CCA application for a chain of N vehicles when the technology penetration rate is not 100%. We consider the inter-vehicle spacing is normally distributed and each vehicle $C_i,\ i \in \{1,...,N\}$, moves at constant velocity V_i. Vehicles drive in convoy (see Figure 1), reacting to the first collision of another car, C_0, according to two possible schemes: starting to brake because of a previously received warning message transmited by a collided vehicle (if the vehicle is equipped with CCA technology) or starting to decelerate after noticing a reduction in the speed of the vehicle immediately ahead (if the vehicle under consideration is not equipped with CCA technology).

With this model the final outcome of a vehicle depends on the outcome of the preceding vehicles. Therefore, the collision model is based on the construction of the following probability tree. We consider an initial state in which no vehicle has collided.

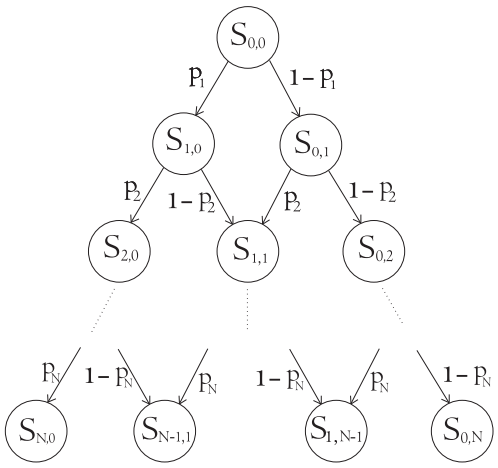

Fig. 2. Probability tree diagram that defines the model. $S_{i,j}$ represents the state with i collided vehicles and j successfully stopped vehicles.

Once the danger of collision has been detected, the first vehicle in the chain C_1 (immediately after the leading one) may collide or stop successfully. From both of these states two possible cases spring as well, that is either the following vehicle in the chain C_2 may collide or stop successfully. And so on until the last vehicle in the chain. At the last level of the tree we have $N+1$ possible outcomes (final outcomes) which represent the number of collided vehicles in the chain, that is, from 0 to N collisions (Figure 2).

The transition probability between the nodes of the tree is the probability of collision of the corresponding vehicle in the chain p_i (or its complementary). These probabilities are calculated recursively, regarding different kinematic parameters, as the average velocity of the vehicles in the chain (used to compute the distance to stop), the average inter-vehicle distance and the driver's reaction time, among others.

We start calculating the collision probability of the nearest to the incidence vehicle, C_1. The position of C_i when it starts to decelerate is normally distributed with mean $\mu_i = -i * d$ and standard deviation $\sigma = d/2$, where d is the average inter-vehicle distance. Vehicle C_1 will collide if and only if the distance to C_0 is less than the distance that it needs to stop, D_s, so its collision probability is given by:

$$p_1 = 1 - \int_{-\infty}^{-L-D_s} f(x;\mu_1,\sigma)\, dx\,, \tag{1}$$

where L is the average vehicle length and $f(x;\mu,\sigma)$ is the probability density function of the normal distribution with mean μ and standard deviation σ.

To compute the collision probability of the second vehicle we will use the average position of the first vehicle when it has stopped (either by collision or successfully stop). This average position is determined by:

$$\overline{X_1} = \int_{-\infty}^{-L} x \cdot f(x;\mu_1+D_s,\sigma)\, dx \;+\; (-L)\cdot \int_{-L}^{+\infty} f(x;\mu_1+D_s,\sigma)\, dx\,. \tag{2}$$

The second term of the sum means that the vehicle cannot cross the position $-L$ when it collides, since we are assuming that when a vehicle collides it stops instantly at the point of collision.

Once we have obtained $\overline{X_1}$ we can compute p_2, and recursively we can obtain all the collision probabilities:

$$p_i = 1 - \int_{-\infty}^{\overline{X_{i-1}}-L-D_s} f(x;\mu_i,\sigma)\, dx, \quad i = 2,\ldots,N\,, \tag{3}$$

where

$$\overline{X_i} = \int_{-\infty}^{\overline{X_{i-1}}-L} x\cdot f(x;\mu_i+D_s,\sigma)\, dx + (\overline{X_{i-1}}-L)\cdot \int_{\overline{X_{i-1}}-L}^{+\infty} f(x;\mu_i+D_s,\sigma)\, dx,\ \ i=2,\ldots,N\,. \tag{4}$$

We want to remark that this model for the collision probabilities is a preliminary approximation and does not describe realistically the collision process. However, the method to compute the probabilities of the path outcomes is independent of the correctness or accuracy of the transition probabilities used, and the goal of this paper is to evaluate the benefits of parallelization for this technique to compute the average number of accidents. An improved model for the transition probabilities can be found in [28].

Let us note how every path in the tree from the root to the leaves leads to a possible outcome involving every vehicle in the chain. The probability of a particular path is the product of the transition probabilities that belongs to the path. Since there are multiple paths that lead to the same final outcome (leaf node in the tree), the probability of that outcome will be the sum of the probabilities of every path reaching it.

In order to compute the probabilities of the final outcomes, we can construct a Markov chain whose state diagram is shown in Figure 2 and is based on the previously discussed probability tree. It is a homogeneous Markov chain with $\frac{(N+1)(N+2)}{2}$ states,

$$(S_{0,0},\ S_{1,0},\ S_{0,1},\ldots,\ S_{N,0},\ S_{N-1,1},\ldots,\ S_{1,N-1},\ S_{0,N})\,. \tag{5}$$

The transition matrix P of the resulting Markov chain is a square matrix of dimension $\frac{(N+1)(N+2)}{2}$, which is a sparse matrix, since from each state it is only possible to move to two of the other subsequent states.

Then, we need to compute the probabilities of going from the initial state to each of the $N+1$ final states in N steps, which are given by matrix P^N. Therefore, the final outcome probabilities are the last $N+1$ entries of the first row of the matrix P^N.

Let Π_i be the probability of reaching the final outcome with i collided vehicles, that is, state $S_{i,N-i}$. We obtain the average of the total number of accidents in the chain using the weighted sum:

$$N_{acc} = \sum_{i=0}^{N} i\cdot\Pi_i\,. \tag{6}$$

Our purpose is to evaluate the functionality of the CCA system depending on the current penetration rate of this technology. So that, we have to solve the model assuming different technology penetration ratios. This assumption implies that we have to

Table 1. Number of combinations of $N = \{10, 20, 30\}$ vehicles with and without CCA technology.

CCA%	10 veh.	20 veh.	30 veh.
0%	1	1	1
10%	10	190	4060
20%	45	4845	593775
30%	120	38760	14307150
40%	210	125970	86493225
50%	252	184756	155117520

calculate the number of collisions once for each of the possible combinations in the chain of vehicles equipped with and without CCA technology, that is,

$$\binom{N}{m} = \frac{N!}{(N-m)!\, m!}\,, \tag{7}$$

where N is the total number of vehicles in the chain and m is the number of vehicles equipped with the CCA technology. It is worth to notice that the number of combinations for m vehicles set with CCA technology and $N-m$ without it is the same that for $N-m$ vehicles with CCA and m without it. Therefore, in order to analyze the computation time, we solve the model varying the CCA penetration rate between 0% and 50%, since the rest of cases are computationally (but not numerically) identical. As we can see in Table 1, the number of combinations grows quickly by an increase on the CCA penetration rate as well as by an increase on the number of vehicles.

In addition to that, we also aim at evaluating the impact on the number of accidents of the inter-vehicle distance d, varying this parameter in a wide range.

5 Implementation

In this section we firstly introduce the algorithm for the model implementation (Algorithm 1) and then, we explain the method we have used to parallelize it.

Examining the algorithm we can make the following observations:

1. The iterations of the *for* loop that covers the number of *Combinations* resulting from the CCA technology penetration rate are independent for each other, so they can be executed in parallel by different threads.
2. The same occurs with the *for* loop that covers the *RangeOfDistances* (for the inter-vehicle spacing) to be evaluated.
3. Since the collision probabilities of the vehicles in the platoon is computed recursively, each iteration of the *for* loop that considers each vehicle in the chain needs the results of the preceding iteration, so this loop should be executed sequentially.
4. To obtain the first row of matrix P^N we have to multiply N times a vector of dimension $\frac{(N+1)(N+2)}{2}$ by a matrix of dimension $\frac{(N+1)(N+2)}{2} \times \frac{(N+1)(N+2)}{2}$. The vector-matrix multiplication can be also parallelized so that each thread executes the multiplication of the vector by part of the matrix columns. However, the N multiplications should be done one after the other, that is, sequentially.

Algorithm 1 Computation of the number of collisions in a chain of vehicles

for all *comb* in *Combinations* **do**
  for all d in *RangeOfDistances* **do**
    for $i = 1$ **to** N **do**
      $p_i = f(p_{i-1}, comb, d, i, veloc, reactTime)$
    end for
    for $j = 0$ **to** N **do**
      $\Pi_j = P^N(1, \frac{(N+1)(N+2)}{2} - j)$
    end for
    $N_{acc} = \sum_{j=0}^{N} j \cdot \Pi_j$
  end for
end for

Table 2. Resulting programs with different parallelized tasks. X means that the corresponding parallelization takes place.

Program	A	B	C
Program 1			
Program 2	×		
Program 3		×	
Program 4			×
Program 5	×	×	
Program 6	×		×
Program 7		×	×
Program 8	×	×	×

For the sake of clarity, we will parallelize the following tasks:

- A: Vector-Matrix multiplication.
- B: Average inter-vehicle distance variation.
- C: Technology penetration rate variation.

Next, we will combine the different parallelized tasks (see Table 2) and execute the resulting programs in order to assess the actual improvement obtained from each one.

6 Results

In this section we summarize the results obtained by executing the programs shown in Table 2 in a node of the Arabi cluster. We have used 2, 4 and 8 processors in order to assess the improvement on the execution time achieved by each one.

The parameters used to execute the model are the following:

- CCA penetration rate: $0\% - 50\%$, in 10% steps.
- Average inter-vehicle distance: $6 - 70$ m, in 1 meter steps.
- Number of vehicles: 20 vehicles.

Table 3. Execution times in minutes and speedup (SU) for each program using 2 processors.

	0%		10%		20%		30%		40%		50%	
	Time	**SU**	**Time**	**SU**	**Time**	**SU**	**Time**	**SU**	**Time**	**SU**	**Time**	**SU**
P1	0.002	1.00	0.307	1.00	7.848	1.00	62.876	1.00	203.896	1.00	297.975	1.00
P2	0.002	1.00	0.247	1.24	6.694	1.17	40.693	1.54	125.658	1.62	188.396	1.58
P3	0.001	2.00	0.175	1.75	4.315	1.82	33.858	1.86	110.142	1.85	159.558	1.87
P4	0.003	0.67	0.147	2.09	3.655	2.15	29.323	2.14	95.671	2.13	157.483	1.90
P5	0.001	2.00	0.173	1.77	4.326	1.81	34.208	1.84	108.542	1.88	161.026	1.85
P6	0.004	0.50	0.167	1.84	4.227	1.86	33.009	1.90	107.534	1.90	156.688	1.90
P7	0.002	1.00	0.167	1.84	4.176	1.88	32.771	1.92	106.119	1.92	156.433	1.90
P8	0.002	1.00	0.168	1.83	4.226	1.86	32.962	1.91	107.422	1.90	158.509	1.88

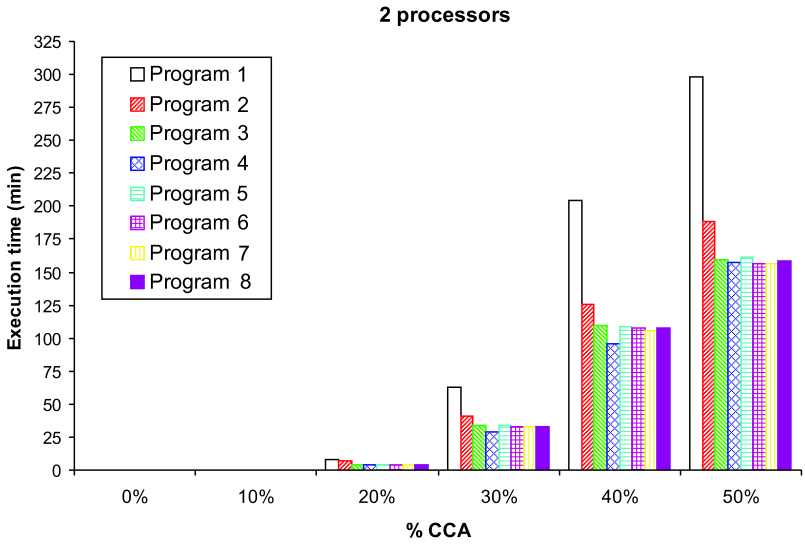

Fig. 3. Execution times in minutes for each program using 2 processors.

– Average velocity: 33 m/s.
– Average driver's reaction time: 1 s.

6.1 Execution with 2 Processors

The computation times resulting from the execution of the eight programs with the selected penetration rates of CCA technology using 2 processors are gathered in Table 3 and illustrated in Figure 3.

Now we focus on the results associated to the 50% CCA penetration rate, since for this value we obtain the highest number of combinations, specifically for a chain of 20

Table 4. Execution times in minutes and speedup for each program using 4 processors.

	0%		**10%**		**20%**		**30%**		**40%**		**50%**	
	Time	**SU**	**Time**	**SU**	**Time**	**SU**	**Time**	**SU**	**Time**	**SU**	**Time**	**SU**
P1	0.002	1.00	0.308	1.00	7.838	1.00	62.653	1.00	203.757	1.00	297.930	1.00
P2	0.001	2.00	0.199	1.55	5.053	1.55	30.676	2.04	94.173	2.16	135.907	2.19
P3	0.001	2.00	0.098	3.14	2.473	3.17	19.488	3.21	59.724	3.41	95.360	3.12
P4	0.004	0.50	0.078	3.95	1.998	3.92	16.072	3.90	51.830	3.93	86.175	3.45
P5	0.002	1.00	0.101	3.05	2.494	3.14	19.933	3.14	63.464	3.21	95.158	3.13
P6	0.005	0.40	0.091	3.38	2.251	3.48	18.013	3.48	59.810	3.40	89.064	3.34
P7	0.004	0.50	0.089	3.46	2.232	3.51	17.754	3.53	57.699	3.53	85.988	3.46
P8	0.003	0.67	0.090	3.42	2.245	3.49	17.926	3.49	59.453	3.43	88.422	3.37

vehicles we obtain a total of 184756 combinations. Therefore, it is for this particular penetration rate when we obtain a higher execution time and it can be considered as the critical case in terms of the solving time.

The sequential program (Program 1) lasts a total of 297.975 minutes, that is approximately 5 hours of computation. If we make a comparison among the parallelized programs we conclude that the best result is given by the Program 7, with a computation time of 156.433 minutes, what implies around 2.6 hours of calculation time. It is worth mentioning that Program 7 is built by a combination of the parallelized tasks B and C, parallelizing the *for* loops that cover the range of average inter-vehicle distances and the number of combinations resulting from the technology penetration rate respectively. We obtain thus:

- Sequential time (P1): 297.975 minutes.
- Parallel time (P7): 156.433 minutes.

The achieved speedup (P1/P7) is 1.9, which implies an improvement of around 47.5% referred to the execution time.

6.2 Execution with 4 Processors

The computation times resulting from the execution of the eight programs with the selected penetration rates of CCA technology using 4 processors are presented in Table 4 and depicted in Figure 4.

When the CCA penetration rate equals the 50% we reach the highest computational load. So we also analyze the results with this penetration rate using 4 processors, focusing on the best and worst execution times achieved. The reference is still the sequential Program 1 with a duration of 297.93 minutes (around 5 hours). If we make a comparison among the parallelized programs we conclude that the best result is given again by the Program 7 with a calculation time of 85.988 minutes (around 1.43 hours). We obtain thus:

- Sequential time (P1): 297.93 minutes.
- Parallel time (P7): 85.988 minutes.

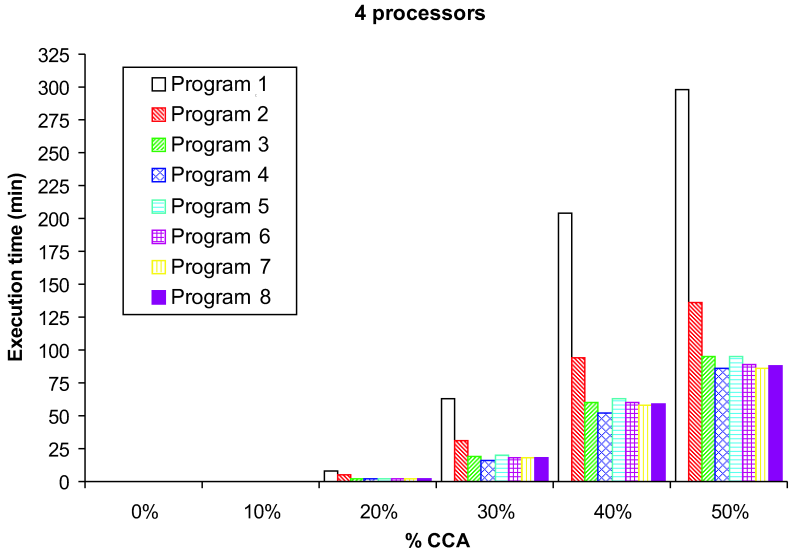

Fig. 4. Execution times in minutes for each program using 4 processors.

Table 5. Execution times in minutes and speedup for each program using 8 processors.

	0%		10%		20%		30%		40%		50%	
	Time	**SU**	**Time**	**SU**	**Time**	**SU**	**Time**	**SU**	**Time**	**SU**	**Time**	**SU**
P1	0.002	1.00	0.308	1.00	7.844	1.00	62.695	1.00	203.416	1.00	296.691	1.00
P2	0.003	0.67	0.193	1.59	4.610	1.70	26.578	2.36	78.644	2.58	117.415	2.53
P3	0.001	2.00	0.067	4.60	1.767	4.44	13.634	4.60	45.213	4.50	62.572	4.74
P4	0.008	0.25	0.047	6.55	1.155	6.79	9.310	6.73	32.142	6.33	54.165	5.48
P5	0.002	1.00	0.071	4.34	1.739	4.51	15.125	4.14	45.858	4.43	62.572	4.74
P6	0.005	0.40	0.055	5.60	1.258	6.23	10.158	6.17	35.275	5.76	54.006	5.49
P7	0.008	0.25	0.054	5.70	1.232	6.37	10.041	6.24	34.800	5.84	50.402	5.89
P8	0.007	0.28	0.051	6.04	1.248	6.28	10.143	6.18	35.376	5.75	53.031	5.59

The achieved speedup is 3.46, which implies an improvement of around 71.1% referred to the execution time.

6.3 Execution with 8 Processors

The computation times resulting from the execution of the eight programs with the selected penetration rates of CCA technology using 8 processors are gathered in Table 5 and illustrated in Figure 5.

Finally we analyze what happens if we use 8 processors to solve the problem. Once more, we obtain for the parallelized Program 7 the least computation time, 50.402 minutes with a 50% CCA penetration rate. So if we compare this result with the execution

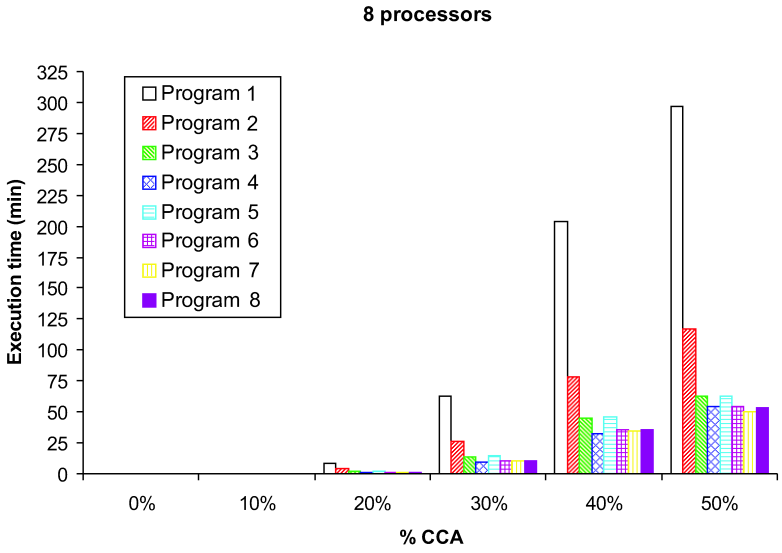

Fig. 5. Execution times in minutes for each program using 8 processors.

time of the sequential program we obtain an improvement of the 83%, that is, a speedup factor of 5.89.

6.4 Results Discussion

In conclusion, on the one hand, we have achieved an improvement of 83% in the computation time of the most complex case, what can be considered as a pretty much outstanding improvement. On the other hand, if we compare the best execution times between the two technical extremes under study, that is the use of 2 or 8 processors belonging to the shared nodes architecture in the Arabi cluster, we reach to an improvement of 67.78%, which implies an upwards trend with increasing the number of processors, as expected. Moreover, we can observe that those programs including the parallelization of task C, which implies an acceleration on the loop varying the CCA technology penetration rate, are the fastest ones. Nevertheless, the results obtained from Program 2 show that the improvement achieved parallelizing only the vector-matrix multiplication (task A) is already significant, reaching 60.4% using 8 processors.

Analyzing the speedup for programs 7 and 8 it surprises that P7, with two parallelized tasks, wins P8 including one more task. But this is a common fact in parallel computing due to load balancing and synchronization overhead [29]. This explains also that all programs including parallelized task C have similar execution times, since this is the heaviest computational task and outshines the improvement derived from the A and B tasks parallelization.

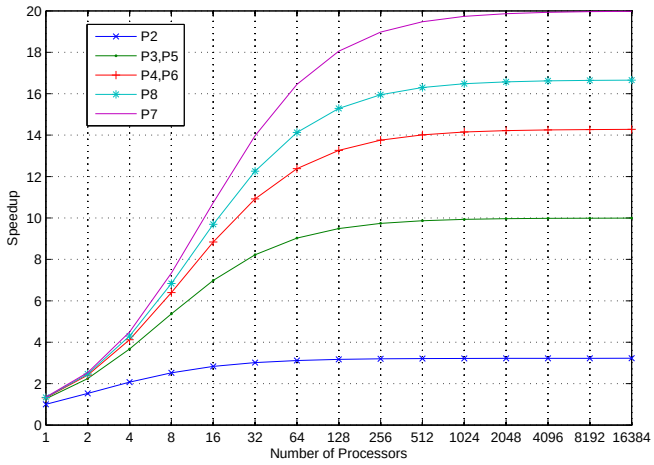

Fig. 6. Theoretical speedup limits calculated from Amdahl's law.

Let us compare now the obtained results for the Program 7, the one with the best execution times, centering on the 50% CCA penetration rate, since as we already mentioned, this is the heaviest option in terms of computational load. We find out an inverse relationship between computation time and the number of processors in use, since when we duplicate the number of processors the execution time of Program 7 is reduced almost to a half. Specifically, the speedup achieved passing from 2 to 4 processors is 1.82, and from 4 to 8 processors, 1.7. However, this speedup is limited according to Amdahl's law [30]. We have calculated for each program the theoretical speedup obtained from this law, as depicted in Figure 6.

Amdahl's law states that if α is the proportion of a program that can be made parallel then the maximum speedup, SU, that can be achieved by using n processors is:

$$SU = \frac{1}{(1-\alpha)+\frac{\alpha}{n}} \,. \tag{8}$$

We can estimate α by using the measured speedup SU on a specific number of processors sn as follows:

$$\alpha_{estimated} = \frac{\frac{1}{SU}-1}{\frac{1}{sn}-1} \,. \tag{9}$$

The results show that for Program 2 the speedup obtained with 8 processors is almost the limit for it, but the speedup for Program 7 can still grow up to 20, which implies reducing the execution time to less than 15 minutes.

Unfortunately, we have not been able to check how the results of Amdahl's law approach to reality. We tried to execute the Program 7 in the Superdome Ben, but executing it using 32 cores the time consumed was much higher than using 2 cores in a

node of the cluster. It is owing to the computing speed (819 Gflops in the Superdome and 9.72 Tflops in the cluster).

As an alternative, we tried using MPI (Message Passing Interface Standard) [31] in order to execute our programs using different nodes of the cluster simultaneously. However, we encountered the problem of an excessive memory requirement, due to the need to replicate data across processes, and consequently we failed in the execution of the programs by this way too.

7 Conclusions and Outlook

Thanks to OpenMP parallelization techniques running under a supercomputing shared memory environment we succeded to evaluate the perfomance of a CCA application at different stages of technology deployment. To conclude, we were able to solve a program with a sequential execution time of 297.975 minutes in only 50.402 minutes.

Regarding the problems we have encountered, as future work, we aim to improve our analytical model, trying to reduce as possible the computational and memory costs. We are also facing similar tasks to improve the efficiency of the VANET simulation environments we are using in order to validate our mathematical analyses.

Acknowledgments. This research has been supported by the MICINN/FEDER project grant TEC2010-21405-C02-02/TCM (CALM) and Fundación Séneca RM grant 00002/CS/08 FORMA. It is also developed in the framework of "Programa de Ayudas a Grupos de Excelencia de la Región de Murcia, de la Fundación Séneca, Agencia de Ciencia y Tecnología de la RM". J. García-Haro acknowledges personal grant PR2009-0337 (MICINN/SEU/DGU Programa Nacional de Movilidad de Recursos Humanos de Investigación). J. B. Tomas-Gabarrón thanks the Spanish MICINN for a FPU (REF AP2008-02244) pre-doctoral fellowship. C. García-Costa acknowledges the Fundación Seneca for a FPI (REF 12347/FPI/09) pre-doctoral fellowship. We also want to express our gratitude to Rocío Murcia-Hernández for her programming assistance.

References

1. Tomas-Gabarron, J.B., Egea-Lopez, E., Garcia-Haro, J., Murcia-Hernandez, R.: Performance evaluation of a CCA application for VANETs using IEEE 802.11p. In: IEEE International Conference on Communications Workshops (ICC), pp. 1-5 (2010)
2. The official webpage of the Scientific Park of Murcia, `http://www.cesmu.es`.
3. Murcia-Hernandez, R., Garcia-Costa, C., Tomas-Gabarronm, J.B., Egea-Lopez, E., Garcia-Haro, J.: Parallelization of a mathematical model to evaluate a CCA application for VANETs. In: 1st International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), The Netherlands (2011)
4. Barney, B.: Introduction to Parallel Computing, `https://computing.llnl.gov/tutorials/parallel_comp/`, Lawrence Livermore National Laboratory, USA (2011)
5. Liu, S., Zhang, Y., Sun, X., Qiu, R.: Performance Evaluation of Multithreaded Sparse Matrix-Vector Multiplication using OpenMP. In: IEEE International Conference on High Performance Computing and Communications, pp. 659-665 (2009)

6. Kotakemori, H., Hasegawa, H., Kajiyama, T., Nukada, A., Suda, R., Nishida, A.: Performance Evaluation of Parallel Sparse Matrix-Vector Products on SGI Altix3700. In: OpenMP Shared Memory Parallel Programming. LNCS, pp. 153-163 (2008)
7. Goumas, G., Kourtis, K., Anastopoulos, N., Karakasis, V., Koziris, N.: Performance evaluation of the sparse matrix-vector multiplication on modern architectures. In: The Journal of Supercomputing, vol. 50, no. 1, pp. 36-77 (2009)
8. Williams, S., Olikera, L., Vuducc, R., Shalfa, J., Yelicka, K., Demmel, J.: Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In: Parallel Computing, vol. 35, no. 3, pp. 178-194 (2009)
9. Ning, Z., Jung, Y., Jin, Y., Kim, K.: Route Optimization for GPSR in VANET. In: IEEE International Advance Computing Conference, pp. 569-573 (2009)
10. Wisitpongphan, N., Fan Bai, Mudalige, P., Tonguz, O.K.: On the Routing Problem in Disconnected Vehicular Ad-hoc Networks. In: IEEE International Conference on Computer Communications, pp. 2291-2295 (2007)
11. Du, S., Liu, F.Q., Xu, S.Z., Wang, X.H.: On-demand power control algorithm based on vehicle ad hoc network. In: Jisuanji Gongcheng/ Computer Engineering, vol. 35, no. 16, pp. 97-100 (2009)
12. Fasolo, E., Zanella, A., Zorzi, M.: An effective broadcast scheme for alert message propagation in vehicular ad hoc networks. In: IEEE International Conference on Communications, pp. 3960-3965 (2006)
13. Li, L.J., Liu, H.F., Yang, Z.Y., Ge, L.J., Huang, X.Y.: Broadcasting methods in vehicular ad hoc networks. In: Ruan Jian Xue Bao (Journal of Software), vol. 21, no. 7, pp. 1620-1634 (2010)
14. Harri, J., Filali, F., Bonnet, C.: Mobility models for vehicular ad hoc networks: a survey and taxonomy. In: IEEE Communications Surveys & Tutorials, vol. 11, no. 4, pp. 19-41 (2009)
15. Djenouri, D., Nekka, E., Soualhi, W.: Simulation of mobility models in vehicular ad hoc networks. In: Proceedings of First ICST International Conference on Ambient Media and Systems (2008)
16. Abboud, K., Weihua Z.: Modeling and analysis for emergency messaging delay in vehicular ad hoc networks. In: IEEE Global Telecommunications Conference, pp. 1-6 (2009)
17. Fukuyama, J.: A delay time analysis for multi-hop V2V communications over a linear VANET. In: IEEE Vehicular Networking Conference, pp. 1-7 (2009)
18. Prasanth, K., Duraiswamy, K., Jayasudha, K., Chandrasekar, C.: Minimizing end-to-end delay in Vehicular Ad Hoc Network using Edge Node Based Greedy Routing. In: First International Conference on Advanced Computing, pp. 135-140 (2009)
19. Khabazian, M., Ali, M.: A performance modeling of connectivity in vehicular ad hoc networks. In: IEEE Transactions on Vehicular Technology, vol. 57, no. 4, pp. 2440-2450 (2008)
20. Xie, B., Xiao, X.Q.: Survivability model for vehicular Ad-Hoc network based on Markov chain. In: Jisuanji Yingyong/ Journal of Computer Applications (2008)
21. Glimm, J., Fenton, R.E.: An Accident-Severity Analysis for a Uniform-Spacing Headway Policy. In: IEEE Transactions on Vehicular Technology, vol. 29, no. 1, pp. 96-103 (1980)
22. Touran, A., Brackstone, M.A., McDonald, M.: A collision model for safety evaluation of autonomous intelligent cruise control. In: Accident Analysis and Prevention, vol. 31, no. 5, pp. 567-578 (1999)
23. Kim, T., Jeong, H.Y.: Crash Probability and Error Rates for Head-On Collisions Based on Stochastic Analyses. In: IEEE Transactions on Intelligent Transportation Systems, vol. 11, no. 4, pp. 896-904 (2010)
24. Cook, W., Rich, J.L., A parallel cutting-plane algorithm for the vehicle routing problem with time windows. Technical Report TR99-04, Department of Computational and Applied Mathematics, Rice University (1999)

25. Ghiani, G., Guerriero, F., Laporte, G., Musmanno, R.: Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. In: European Journal of Operational Research, vol. 151, no. 1, pp. 1-11 (2003)
26. Bouthillier, A.L., Crainic, T.G.: A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. In: Computers & Operations Research, vol. 32, no. 7, pp. 1685-1708 (2005)
27. Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., Menon, R.: Parallel Programming in OpenMP. Morgan Kaufmann, (2001)
28. Garcia-Costa, C., Egea-Lopez, E., Tomas-Gabarron, J.B., Garcia-Haro, J.: A stochastic model for chain collisions of vehicles equipped with vehicular communications. In: IEEE Transactions on Intelligent Transportation Systems, in press. DOI: 10.1109/TITS.2011.2171336.
29. The OpenMP official webpage, `http://openmp.org`.
30. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the spring joint computer conference (1967)
31. The MPI official webpage, `http://www.mpi-forum.org/`.