PARALLELIZATION OF A MATHEMATICAL MODEL TO EVALUATE A CCA APPLICATION FOR VANETS

Rocío Murcia-Hernández, Carolina García-Costa, Juan Bautista Tomás-Gabarrón,
Esteban Egea-López and Joan García-Haro
Department of Information and Communications Technologies, Technical University of Cartagena (UPCT)
Plaza del Hospital 1, Cartagena, Spain
{*rocio.murcia, carolina.garcia, juanba.tomas, esteban.egea, joang.haro*}*@upct.es*

Keywords: OpenMP, VANET, Supercomputing, Cooperative/Chain collision avoidance application.

Abstract: Vehicular Ad-hoc Networks (VANET) are currently becoming not only an extremely important factor for vehicles engineering development but also a key issue for improving 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. We have developed an analytical model to compute the average number of accidents in a platoon of vehicles. However, due to the model structure, when the CCA technology penetration rate is taken into account, the increase in the number of operations of the analytical model is such that the sequential computation of a numerical solution is no longer feasible. In this paper, with the goal in mind of reducing computation time, we show how we have implemented and parallelized our analytical model so as a solution can be achieved, what is conducted using the OpenMP parallelization techniques under a supercomputing shared memory environment.

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 (Tomas-Gabarron et al., 2010) 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, where each vehicle C_i, $i \in 1 \ldots N$, moves at constant velocity. The leading vehicle, C_0, 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 situ-

ation, 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 parallelization techniques such as OpenMP for solving those computational cases considered as unapproachable by means of sequential procedures.

Additionally, we execute our programs in the Ben-Arabi Supercomputing environment (FPCMur, 2011), 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, therefore we succeed to evaluate the CCA application thoroughly.

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. Conclusions and future work are remarked in Section 7.

2 RELATED WORK

So far, most typical high performance computing (HPC) problems focused either on those fields related with the Grand Challenges defined as fundamental problems in science and engineering or directed toward Web search databases (Barney, 2010). 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 (Liu et al., 2009). In general, we find quite a lot of done work in the field of sparse matrix-vector multiplications using parallelization techniques (Kotakemori et al., 2008; Goumas et al., 2009; Williams et al., 2009). 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 (Ning et al., 2009; Wisitpongphan et al., 2007), the broadcasting methods (Du et al., 2009; Fasolo et al., 2006; Li et al., 2010), the mobility of vehicles (Harri et al., 2009; Djenouri et al., 2008) and the communication delay time (Abboud and Zhuang, 2009; Fukuyama, 2009; Prasanth et al., 2009). Other related VANET issues have been studied as well, like network connectivity (Khabazian and Mehmet Ali, 2008), or survivability (Xie and Xiao, 2008). 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 (Glimm and Fenton, 1980; Touran and Brackstone, 1999; Kim and Jeong, 2010).

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 (Cook and Rich, 1999; Ghiani and Guerriero, 2003; Bouthillier and Crainic, 2005).

Therefore and 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 to address the benefits of using parallelization techniques in the VANET arena.

3 SUPPORTING TOOLS

3.1 The OpenMP Technique

OpenMP is a well-known open standard for providing parallelization mechanisms to multiprocessors with shared memory (Chandra et al., 2001). 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 me-

mory programmers with a simple and flexible interface for parallel applications developing 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 this way the execution time.

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 $SX2000$ 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 respectively to compare the different execution times results. Taking into account that we are not allowed to combine the usage of processors from different nodes, since we are using a shared memory parallelization technique.

Next we summarize the cluster technical features:

- 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.

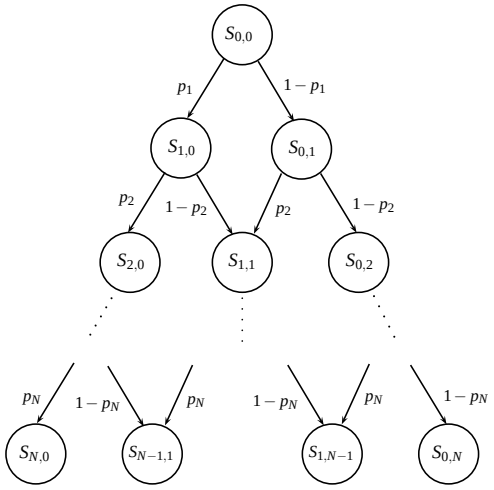

Figure 1: Probability tree that defines the model. $S_{i,j}$ represents the state with i collided vehicles and j successfully stopped vehicles.

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%. Vehicles drive in convoy, reacting to the first collision of another car 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. 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 1).

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 aver-

age 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. The exact details of this calculation are crucial for our analytical model but they are out of the scope of this paper. Let us remark that it is a recursive operation, that is $p_i = f(p_{i-1})$, and must be done sequentially.

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 1 and is based on the previously discussed probability tree. It is a homogeneous Markov chain with $\frac{(N+1)(N+2)}{2}$ states,

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

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:

$$N_{acc} = \sum_{i=0}^{N} i \cdot \Pi_i. \tag{2}$$

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 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,

$$\binom{N}{m} = \frac{N!}{(N-m)!\,m!}, \tag{3}$$

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.

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

In addition to that, we also aim at evaluating the impact on the number of accidents of the inter-vehicular distance, varying this parameter (*dist*) 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.

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

for all *comb* in *Combinations* **do**
  for all *dist* in *RangeOfDistances* **do**
    for $i = 1$ **to** N **do**
      $p_i = f(p_{i-1}, comb, dist, i, veloc, reactTime)$
    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

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-vehicular 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 a $\frac{(N+1)(N+2)}{2}$ dimension vector by a $\frac{(N+1)(N+2)}{2} \times \frac{(N+1)(N+2)}{2}$ matrix N times. 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.

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

- A: Vector-Matrix multiplication.
- B: Average inter-vehicular 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.

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	×	×	×

6 RESULTS

In this section we execute the programs as discussed in the previous section (shown in Table 2) in a node of the Arabi cluster using 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-vehicular distance: 6 – 70 m, in 1 meter steps.
- Number of vehicles: 20 vehicles.
- 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 2.

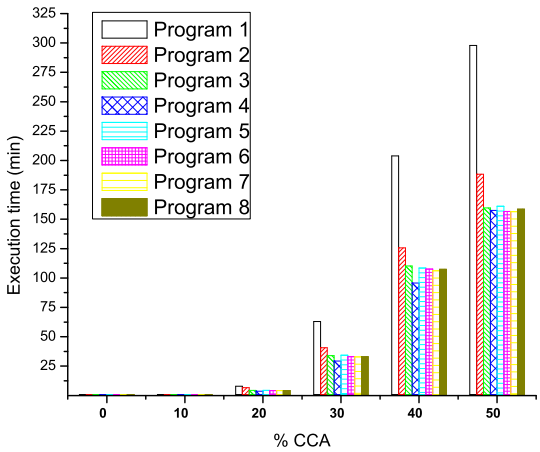

Figure 2: Execution times in minutes for each program using 2 processors.

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 vehicles we obtain a total of 184756 combinations. Therefore,it is for this particular penetration rate case 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-vehicular distances and the number of combinations resulting from the technology penetration rate respectively. We obtain thus:

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

- 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 3.

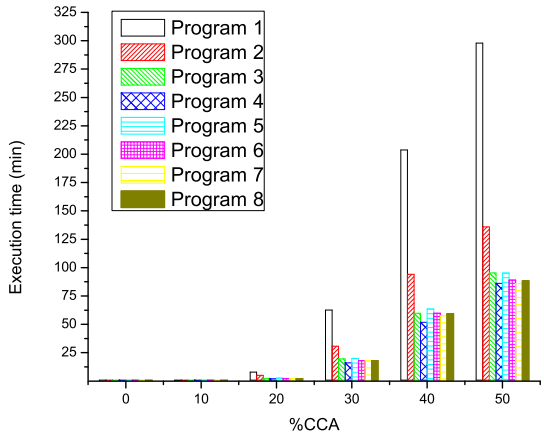

Figure 3: Execution times in minutes for each program using 4 processors.

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.

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 4.

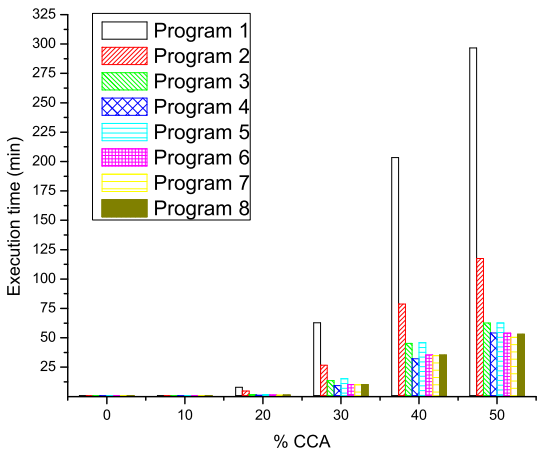

Figure 4: Execution times in minutes for each program using 8 processors.

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

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

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

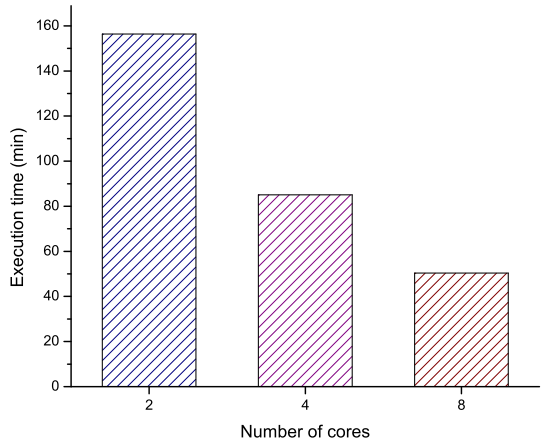

Figure 5: Execution times for Program 7 with 50% of CCA penetration rate using 2, 4 and 8 processors.

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 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 paralellized tasks, wins P8 including one more task. But this is a common fact in parallel computing due to load balancing and synchronization overhead (OpenMP, 2011). 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.

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. These results are depicted in Figure 5. 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 (Amdahl, 1967). 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:

$$SU = \frac{1}{(1-\alpha)+\frac{\alpha}{n}}. \quad (4)$$

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

$$\alpha_{estimated} = \frac{\frac{1}{SU}-1}{\frac{1}{sn}-1}. \quad (5)$$

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. So, as future work, we will try to execute our programs with a higher amount of processors in order to evaluate the system involving a more number of vehicles.

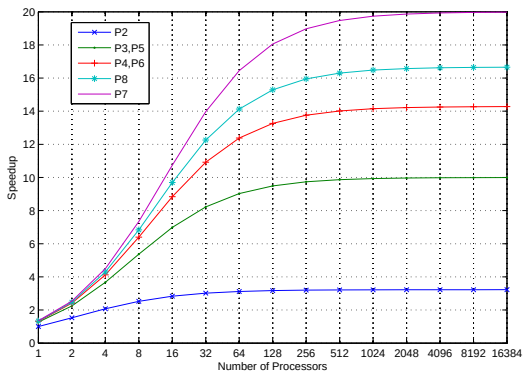

Figure 6: Theoretical speedup limits calculated from Amdahl's law.

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 an execution time of 297.975 minutes in only 50.402 minutes.

As future work, on the one hand we will try to execute our programs with a higher amount of processors in order to evaluate the system involving a bigger number of vehicles. On the other hand, we aim to deepen and improve our analytical model and study further, within this model, the use of parallelization in a supercomputing platform. 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.

ACKNOWLEDGEMENTS

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. Garcia-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. Garcia-Costa acknowledges the Fundación Seneca for a FPI (REF 12347/FPI/09) pre-doctoral fellowship.Rocío Murcia-Hernández acknowledges the personal grant she is enjoying belonging to the project FORMA and specially the Supercomputating center of the Scientific Park Foundation in Murcia (Spain).

REFERENCES

Abboud, K. and Zhuang, W. (2009). Modeling and analysis for emergency messaging delay in vehicular ad hoc networks. In *IEEE Global Telecommunications Conference*.

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the spring joint computer conference*.

Barney, B. (2010). Introduction to parallel computing. https://computing.llnl.gov/tutorials/parallel_comp/.

Bouthillier, A. L. and Crainic, T. G. (2005). A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. In *Computers & Operations Research*.

Chandra, R., Dagum, L., and et al. (2001). *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers.

Cook, W. and Rich, J. L. (1999). A parallel cutting-plane algorithm for the vehicle routing problem with time windows. In *Computational and Applied Mathematics Rice University*.

Djenouri, D., Nekka, E., and et al. (2008). Simulation of mobility models in vehicular ad hoc networks. In *Proceedings of the 2008 Ambi-Sys workshop on Software Organisation and MonIToring of Ambient Systems*.

Du, S., Liu, F. Q., and et al. (2009). On-demand power control algorithm based on vehicle ad hoc network. In *Jisuanji Gongcheng/ Computer Engineering*.

Fasolo, E., Zanella, A., and et al. (2006). An effective broadcast scheme for alert message propagation in vehicular ad hoc networks. In *IEEE International Conference on Communications*.

FPCMur (2011). Scientific park in murcia, official webpage. http://www.cesmu.es.

Fukuyama, J. (2009). A delay time analysis for multi-hop v2v communications over a linear vanet. In *IEEE Vehicular Networking Conference*.

Ghiani, G. and Guerriero, F. e. a. (2003). Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. In *European Journal of Operational Research*.

Glimm, J. and Fenton, R. E. (1980). An accident-severity analysis for a uniform-spacing headway policy. In *IEEE Transactions on Vehicular Technology*.

Goumas, G., Kourtis, K., and et al. (2009). Performance evaluation of the sparse matrix-vector multiplication on modern architectures. In *The Journal of Supercomputing*.

Harri, J., Filali, F., and et al. (2009). Mobility models for vehicular ad hoc networks: a survey and taxonomy. In *IEEE Communications Surveys & Tutorials*.

Khabazian, M. and Mehmet Ali, M. K. (2008). A performance modeling of connectivity in vehicular ad hoc networks. In *IEEE Transactions on Vehicular Technology*.

Kim, T. and Jeong, H. Y. (2010). Crash probability and error rates for head-on collisions based on stochastic analyses. In *IEEE Transactions on Intelligent Transportation Systems*.

Kotakemori, H., Hasegawa, H., and et al. (2008). Performance evaluation of parallel sparse matrixvector products on sgi altix3700. In *OpenMP Shared Memory Parallel Programming. Lecture Notes in Computer Science*.

Li, L. J., Liu, H. F., and et al. (2010). Broadcasting methods in vehicular ad hoc networks. In *Ruan Jian Xue Bao (Journal of Software)*.

Liu, S., Zhang, Y., and et al. (2009). Performance evaluation of multithreaded sparse matrix-vector multiplication using openmp. In *IEEE International Conference on High Performance Computing and Communications*.

Ning, Z., Jung, Y., and et al. (2009). Route optimization for gpsr in vanet. In *IEEE International Advance Computing Conference*.

OpenMP (2011). Openmp, official webpage. http://openmp.org.

Prasanth, K., Duraiswamy, K., and et al. (2009). Minimizing end-to-end delay in vehicular ad hoc network using edge node based greedy routing. In *First International Conference on Advanced Computing*.

Tomas-Gabarron, J. B., Egea-Lopez, E., and et al. (2010). Performance evaluation of a cca application for vanets using ieee 802.11p. In *IEEE International Conference on Communications Workshops (ICC)*.

Touran, A. and Brackstone, M. A. e. a. (1999). A collision model for safety evaluation of autonomous intelligent cruise control. In *Accident Analysis and Prevention*.

Williams, S., Oliker, L., and et al. (2009). Optimization of sparse matrixvector multiplication on emerging multicore platforms. In *Parallel Computing*.

Wisitpongphan, N., Bai, F., and et al. (2007). On the routing problem in disconnected vehicular ad-hoc networks. In *IEEE International Conference on Computer Communications*.

Xie, B. and Xiao, X. Q. e. a. (2008). Survivability model for vehicular ad-hoc network based on markov chain. In *Jisuanji Yingyong/ Journal of Computer Applications*.