Parallel implementation of a hyperspectral image linear SVM classifier using RVC-CAL

D. Madroñal[1], H. Fabelo[2], R. Lazcano[1], G. M. Callicó[2], E. Juárez[1], C. Sanz[1]

[1]Centre of Software Technologies and Multimedia Systems (CITSEM), Technical University of Madrid (UPM), Spain

[2]Research Institute for Applied Microelectronics (IUMA), University of Las Palmas de Gran Canaria (ULPGC), Spain

{daniel.madronal, raquel.lazcano, eduardo.juarez, cesar.sanz}@upm.es
{hfabelo, gustavo}@iuma.ulpgc.es

ABSTRACT

Hyperspectral Imaging (HI) collects high resolution spectral information consisting of hundreds of bands across the electromagnetic spectrum –from the ultraviolet to the infrared range–. Thanks to this huge amount of information, an identification of the different elements that compound the hyperspectral image is feasible. Initially, HI was developed for remote sensing applications and, nowadays, its use has been spread to research fields such as security and medicine. In all of them, new applications that demand the specific requirement of real-time processing have appear. In order to fulfill this requirement, the intrinsic parallelism of the algorithms needs to be explicitly exploited.

In this paper, a Support Vector Machine (SVM) classifier with a linear kernel has been implemented using a dataflow language called RVC-CAL. Specifically, RVC-CAL allows the scheduling of functional actors onto the target platform cores. Once the parallelism of the classifier has been extracted, a comparison of the SVM classifier implementation using LibSVM –a specific library for SVM applications– and RVC-CAL has been performed.

The speedup results obtained for the image classifier depends on the number of blocks in which the image is divided; concretely, when 3 image blocks are processed in parallel, an average speed up above 2.50, with regard to the RVC-CAL sequential version, is achieved.

Keywords: Hyperspectral Imaging; Support Vector Machine; RVC-CAL; Real-time processing; Parallelism exploitation

1. INTRODUCTION

Hyperspectral images gather a huge amount of information covering hundreds of spectral bands ranging from the infrared to the ultraviolet spectrum. Moreover, Hyperspectral Imaging (HI) technology [1] aims at using them to distinguish the different materials that compound a captured scene and to estimate their distribution [2]. Although HI was originally developed for remote sensing applications, nowadays it is being applied to other research fields as astronomy, security, forensics and medicine [3]-[6].

One of the most extended procedures to identify materials and locate their distribution within a captured scene is the supervised learning classification [7] [8]. This methodology is based on generating a classification model by extracting a series of characteristics from a labeled training dataset composed of pixels whose materials –the so-called classes– are known in advance. Once this classification model has been defined and the characteristics associated to each material have been extracted, the classifier is able to assign a class to a new set of unlabeled pixels.

Within the different methodologies of the supervised learning classification, Support Vector Machines (SVM) present the best performance when the number of training samples is reduced. This classifier has been extensively utilized to analyze hyperspectral images and its good performance has been tested in real-time applications [9].

Related with HI applications, there are several research fields aiming at processing hyperspectral images in real-time which, in turn, mean to classify the captured scenes in a reduced amount of time [10]. To do so, one of the most extended procedures is to exploit the intrinsic parallelism of the algorithms which, in consequence, supports the idea of using the SVM classifier due to its pixel-wise processing independency, i.e. each pixel is classified independently from the rest of the image [11].

In this line, dataflow languages try to ease the parallelization process by separating the implementation of the algorithm functionalities from the communication within the different stages. One example of these languages is known as Reconfigurable Video Coding CAL Actor Language (RVC-CAL) [12]. In this language, the functional part of the system is divided in blocks –or actors–. In particular, these actors are composed by a series of input and output ports and perform some tasks of the system, also known as actions. Once the algorithm is divided in actors, the communication among them is described in a graphical way, connecting each output to its corresponding input. Furthermore, RVC-CAL supports a semi-automatic association of each actor to a specific processing unit to explicitly map the system into different cores and, in consequence, parallelize its execution.

The main contribution of this paper is the implementation and parallelization of the SVM classifier algorithm using the RVC-CAL dataflow language. To do so, first an adaptation of the LibSVM code is performed and, after that, the implementation space offered by the prediction stage of the system is studied.

The rest of the paper is structured as follows: first, the state-of-the-art related with this research is briefly exposed; secondly, Section 3 gathers an explanation of the SVM algorithm where its main stages are described and an analysis of the data dependency is carried out; after that, in Section 4, the different steps of the implementation procedure are detailed; the results obtained during the assessment of the SVM conversion to RVC-CAL and the following parallelization of the system are exposed in Section 5; finally, this paper draws some conclusions extracted during this research.

2. RELATED WORK

2.1 SVM classifiers in HI

As mentioned in Section 1, SVM classifiers are an extended methodology to classify hyperspectral images among the supervised learning classifiers. In this line, in [13], the effectiveness of SVMs with respect to neural networks and K-nearest neighbors method when classifying hyperspectral images is assessed, proving that SVMs are a valid and effective procedure to use in remote sensing applications.

Likewise, in [14], Bazi exposes a series of experiments to improve the efficiency of SVMs in this kind of applications, focusing on finding the minimum number of support vectors generated during the training and, in addition, reducing the security margins of the limits among classes.

Moreover, in order to improve the processing performance of SVM classifiers, there are multiple studies dedicated to parallelize this algorithm. Specifically, in [15] and [16], the SVM algorithm has been implemented in GPU platforms and, as a result, speedups of 9 and 10 have been obtained when comparing with a classical multiclass solver implemented using LibSVM.

2.2 RVC-CAL

Dataflow languages are aimed at reducing the parallelization process complexity of a system. To do so, system specification with the CAL Actor Language (CAL) is based on creating a graph where the blocks, formally known as actors, are independent functional units where specific tasks are carried out. On the other hand, the connections represent the information transmissions performed among actors [17].

In this line, ISO/IEC Motion Pictures Experts Group (MPEG) has standardized the RVC-CAL dataflow language [12]. The specification process in RVC-CAL is divided in two parts. During the first part, the different system functionalities are divided and individually implemented using actors. These actors, in turn, are divided into some actions that perform the functionalities associated to the actor and input and output ports are assigned to the actor. On the other hand, the second step is to implement the communications among the actors using a block diagram or graph where the actors are represented as blocks or nodes and the connections or arcs are associated with the communications among them. Specifically, these communications are based on sending packages of information that receive the name of tokens in this language. Likewise, the size of these tokens is defined for each communication by the implicated actors.

Furthermore, a new tool called ORCC –which stands for Open RVC-CAL Compiler– allows the automatic generation of, for instance, C code from the RVC-CAL specification. By doing so, the portability among platforms of the generated code increases exponentially. Additionally, this tool presents the capability of scheduling the actor execution within the different processing units of the platform which, in turn, completes the parallelization procedure of the system [18].

3. MODELING METHODOLOGY

This section aims at analyzing the different stages of the Support Vector Machine classifier. After each stage analysis, a brief conclusion of the data dependency within the stage is exposed.

3.1 Support Vector Machines

As mentioned in Section 1, this research is aimed at implementing a Support Vector Machine classifier using a dataflow language called RVC-CAL. This kind of classifiers are usually composed of two main stages: first, a training stage where a classification model is generated based on a training set of pixels –pixels whose associated class are known in advance– and, after that, a prediction step where a set of unlabeled pixels is classified using the previously obtained model.

During the first stage, the training set is analyzed to obtain a series of support vectors which, in turn, will define a series of hyperplanes to separate pairs of classes. Specifically, the number of classes existing within the case under study determines the number of hyperplanes that needs to be defined. In particular, the relationship between classes to be distinguished and calculated hyperplanes follows equation (1), where N is the number of classes and H is the number of hyperplanes. For example, 3 hyperplanes are required in three-class problems and 10 in five-class approaches. In consequence, the number of hyperplanes grows as a quadratic function with the number of classes under study.

$$H = \frac{(N-1) * N}{2} \tag{1}$$

As this stage is aimed at configuring the classifier with samples whose classes are known in advance, it can be considered as a previous step of the processing chain. Due to its independency from the data to be classified, the training can be carried out offline.

On the other hand, the prediction stage aims at associating a class to an unlabeled pixel –or set of pixels– based on the classification model previously generated. To do so, when a multi-class approach (i.e. when more than two classes need to be distinguished) is being studied, three stages need to be fulfilled:

- First, a binary classification stage where the distance of the pixel to each hyperplane of the classification model is computed: for each distance, only the support vectors associated to the hyperplane that separates two classes are taken into account. To do so, the kernel theory explained in [19] is applied. The most extended strategy is the linear kernel, which follows equation (2). As can be seen, the distance is computed by a sum of two terms. The first one is an accumulation of dot products of the support vectors $\boldsymbol{x_i}$ belonging to the set S defined by the classification model and the sample to classify, $\boldsymbol{x}$. Each dot product is weigh by the Lagrange multiplier, α_i, and a class index, y_i, associated to each support vector. The second one is the bias b associated to the hyperplane.

$$f(x) = \sum_{i \in S} \alpha_i \cdot y_i(\boldsymbol{x_i} \cdot \boldsymbol{x}) + b \tag{2}$$

- Secondly, a probability estimation step is performed. During this stage, the probability of the pixel to belong to each possible class is calculated. To do so, the distances to each binary classifier are considered as inputs and a series of conditional probabilities are generated using a sigmoid function. After that, a multi-class estimation procedure is carried out to combine the conditional probabilities and to obtain the global class probabilities. Specifically, as explained in [20], this combination of conditional probabilities can be computed using different strategies.

- Finally, a final class needs to be assign to every pixel within the captured scene. There are two main methodologies to do so: the former is to apply a majority rule to the binary classification stage and to select the winning one as the pixel final class; the latter is to analyze the global class probabilities obtained during the multi-class probability estimation stage and to select as the pixel final class the one with the largest probability.

To conclude this analysis, it must be highlighted that the prediction stage is computed for each pixel independently, i.e. this stage can be considered as a pixel-wise procedure where each pixel can be classified independently from the rest of the captured scene.

4. IMPLEMENTATION

This section is organized as follows: first, the database utilized to assess the correct behavior of the SVM system is explained; secondly, the parallelization process of the implementation of the SVM algorithm is detailed; finally, the different networks used to configure the dataflow of the system are explained.

4.1 Database

The hyperspectral images utilized to assess the conducted experiments have been extracted from the HELICoiD project database [10]. These images have been obtained from human brain tissue resected during a neurosurgical procedure at the University Hospital Doctor Negrin of Las Palmas de Gran Canaria.

The scenes have been captured using the VNIR camera of the HELICoiD setup [21], which ranges from 400nm to 1000nm. This setup also gathers a push-broom scanning unit and an illumination system that provides a cold light mounted in a structure for surgical environments.

In this case, two hyperspectral images have been selected to evaluate the SVM behavior. After a preprocessing stage, both images have a spectral resolution of 129 bands; likewise, the first image –Case 1– has a spatial resolution of 377 lines x 329 samples per line (124033 pixels) whilst the second image –Case 2– is composed by 479 lines x 552 samples per line (264408 pixels).

4.2 Model implementation

During this research, a linear kernel SVM that distinguishes among three different classes –healthy tissue, tumor tissue and others– has been implemented. Moreover, the target platform is an Intel Core i5 with 4 cores running at 3.3GHz.

First, as the training step is considered as a configuration stage where the classification models are generated, it has been performed offline. Specifically, a MATLAB® model of the training stage has been implemented using LibSVM [22], which is a well-known library extensively used in SVM implementations.

On the contrary, to implement the prediction stage on RVC-CAL, three different functionalities have been extracted and, in consequence, the classification system has been implemented using three different actors. Likewise, the classification of the scene is performed using a pixel-wise strategy, i.e. one pixel is classified and, once its analysis has finished, the prediction of the next one begins.

Figure 1 graphically shows the sequential system that has been implemented using RVC-CAL. As can be observed, each actor carries out a specific functionality:

- The Reader actor aims at loading the classification model, reading the image and sending both to the Predict actor.
- The Predict actor objective is to classify the scene. First, it receives the model; once it has been correctly configured, it receives the image;, afterwards, the actor carries out the prediction stage of the algorithm. Following the pixel-wise independency of the algorithm, a total of three binary classifiers are computed; Then, the multi-class probability estimation is performed and, once the global probabilities are calculated, the class with the largest probability is selected as the pixel final class.
- Finally, the Display actor receives the prediction results –classes and global probabilities– and writes them into a file.

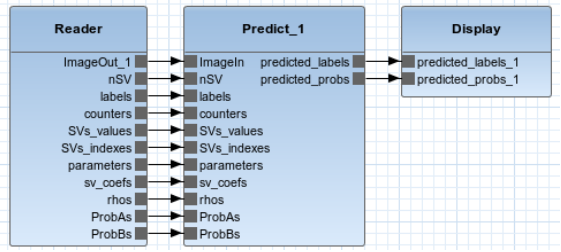

Figure 1. Sequential network on RVC-CAL

4.3 Network mapping

As a final step, the parallelization of the system needs to be explained. To parallelize the system, the prediction stage has been considered as a pixel-wise procedure. Specifically, two different networks have been implemented using at most 3 cores to run the application. As the target environment has a total of 4 available cores, one of them has been reserved for running the operating system to avoid possible interferences.

Figure 2 shows the configuration of the first network for the 3-cores experiment. This distribution aims at parallelizing the prediction step. In this case, the system follows the steps described below:

- First, the Reader actor loads the classification model and broadcasts it. Once the model is sent, it reads the image and, after that, it splits the scene in blocks that will be sent to the Predict actors. Predict_1, Predict_2 and Predict_3.
- The Predict actors are executed in parallel and each actor processes only the piece of image that has received. This parallelization, in theory, divides the prediction step time by the number of actors involved in this computation. Once each actor has finished, the results are sent to the Display actor.
- Finally, when the Display actor receives all the data, it generates two files: one with the classification map and another one with the three probability maps (one per class).

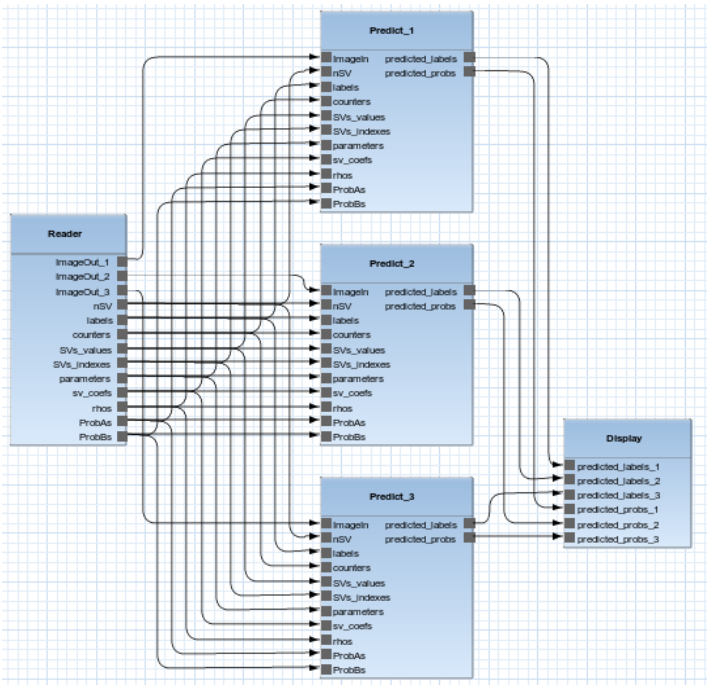

Figure 2. First parallel network on RVC-CAL

On the other hand, the second actor network consists on creating as many processing chains as cores used. Figure 3 shows the network utilized to implement the 3-core classification system. In this case, the system works as follows:

- First, each Reader actor loads the classification model and sends it to its associated Predict actor. After that, only its corresponding portion of the image is loaded to improve the reading time and, once this reading has finished, the image block is sent to the Predict actor.
- The Predict actors work following the same schema than in the previous network: receive the model, receive the image block, process the image block and send the results.
- Finally, each Display actor receives and saves a portion of the probability and the classification maps.

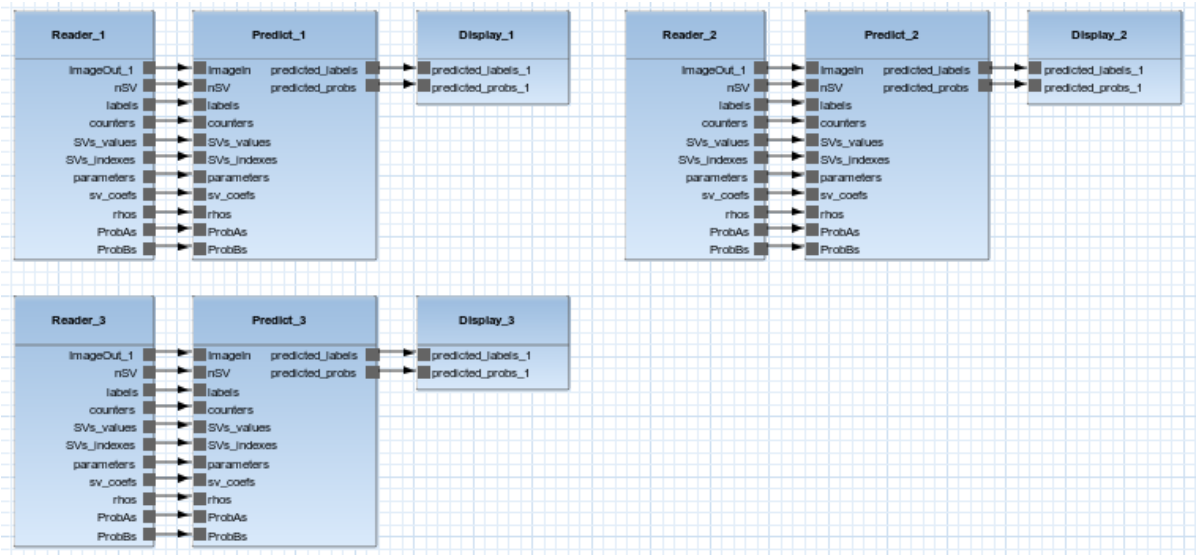

Figure 3. Second parallel network on RVC-CAL

5. RESULTS

During this section, the obtained results are analyzed in terms of execution time. In this case, the analysis has been divided into two parts: first, the adaptation of the LibSVM code to the RVC-CAL language is assessed; secondly, the performances of the parallel implementations of the SVM classifier on RVC-CAL are evaluated.

5.1 RVC-CAL adaptation

In this set of experiments, a comparison between the sequential versions of both LibSVM and RVC-CAL implementations is performed. To measure the execution time, the test-bench is composed by 10 executions of each experiment. Specifically, the execution time has been divided into three components that directly relate to the three functionalities explained in Section 4: read, predict and display. In addition, the total execution time –including the communication between actors– is measured.

Table 1 gathers the average execution times (in seconds) obtained for both implementations and both cases under study. As can be seen, the prediction time has been reduced when the LibSVM code has been adapted to the RVC-CAL language. As a result, a speedup over 4.00 is obtained in both experiments.

Table 1. Comparative of the average execution time and speedup of the sequential version of the classifier using LibSVM and RVC-CAL

Experiment		Read (s)	Predict (s)	Display (s)	Total (s)	Speedup
Case	Version					
1	LibSVM	2.35	72.60	0.29	75.23	1.00
	RVC-CAL	2.32	16.32	0.13	18.78	4.01
2	LibSVM	4.95	385.53	0.31	390.79	1.00
	RVC-CAL	4.93	88.57	0.27	93.84	4.16

Furthermore, it should be highlighted that, as expected, most of the execution time is dedicated to classify the image. Consequently, as the SVM algorithm can be performed using a pixel-wise methodology, these results support the idea of parallelizing the prediction task by dividing the image into blocks. It is also worth noting that, as the image in Case 2 is larger than the one in Case 1, the second experiment requires more time to be fulfilled than the first one.

5.2 Parallel implementation

Once the SVM algorithm has been adapted to the RVC-CAL language, the parallelization of the classifier is evaluated. In this case, two different networks –explained in detail in Section 4– have been analyzed. On the one hand, the first approach presents a parallelization of the prediction procedure. On the other hand, the second one implements a full processing chain for each image block to be processed in parallel, i.e. all the procedures of the system have been parallelized.

Table 2 shows the execution time (in seconds) obtained using the first network as a function of the number of cores employed. As expected, the reading and displaying times remain constant whilst the classification time is almost divided by the number of cores involved in the prediction stage. Furthermore, analyzing the global execution time, speedups of 2.27 and 2.25 are obtained for Case 1 and Case 2, respectively. Ideally, this speedup should be 3 but, as the reading and the display functionalities are performed by only one actor, the image transmission and the result reception are carried out sequentially. Finally, it should be highlighted that, if the prediction time is individually analyzed, the speedups obtained using 3 actors are 2.86 and 2.44 for Case 1 and Case 2, respectively (not shown in Table 2).

Table 2. Average execution time and speedup for Case 1 and Case 2 of the first network implementation of the SVM classifier

Experiment		**Read (s)**	**Predict (s)**	**Display (s)**	**Total (s)**	**Speedup**
Case	**Cores**					
1	1	2.32	16.32	0.13	18.78	1.00
	2	2.35	8.48	0.13	10.96	1.71
	3	2.46	5.69	0.13	8.27	2.27
2	1	4.93	88.57	0.27	93.84	1.00
	2	4.96	44.32	0.26	49.54	1.89
	3	5.09	36.22	0.28	41.59	2.25

The second approach implements an independent processing chain for each image block. Specifically, in this case, all the steps of the processing chain are executed in parallel and, in addition, the transmissions among actors are also performed in parallel.

Table 3 joins the execution times obtained during this experiment. As can be seen, as expected, the execution time of each stage is reduced with the number of active cores. In this case, speedups of 2.89 and 2.50 are obtained for Case 1 and Case 2, respectively, when using 3 independent processing chains.

Table 3. Average Execution time and speedup for Case 1 and Case 2 of the second network implementation of the SVM classifier

Experiment		**Read (s)**	**Classify (s)**	**Display (s)**	**Total (s)**	**Speedup**
Case	**Cores**					
1	1	2.32	16.32	0.13	18.78	1.00
	2	1.17	8.45	0.07	9.68	1.94
	3	0.80	5.66	0.07	6.50	2.89
2	1	4.93	88.57	0.27	93.84	1.00
	2	2.47	44.45	0.24	47.04	1.99
	3	1.70	36.06	0.16	37.56	2.50

To summarize the results obtained during the implementation space exploration, Figure 4 offers a graphical comparison of the total execution times achieved when parallelizing the SVM algorithm using RVC-CAL. Specifically, Case 1 is represented on the upper side, while the lower side displays the results obtained from Case 2. Additionally, the evolution of the speedup is represented. As can be seen, due to the parallelization of the whole chain instead of only parallelizing the prediction step, the improvement achieved with the second approach is 27% and 11% better than that obtained with the first network.

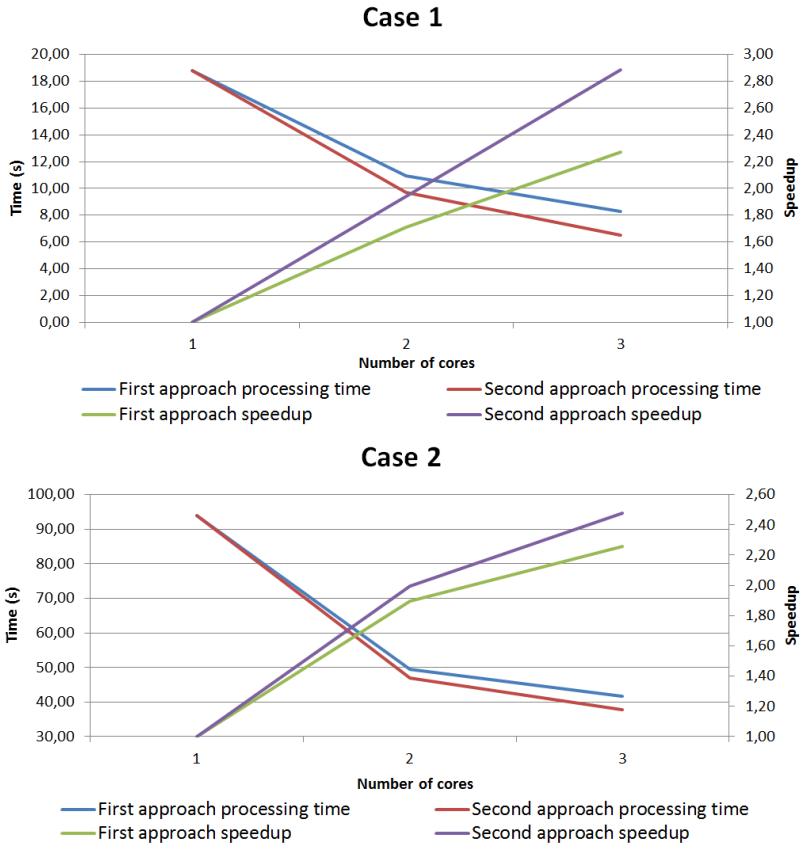

Figure 4. Total processing time and speedup as a function of the number of cores for Case 1 (up) and Case 2 (down)

6. CONCLUSION

During this research, an exploitation of the intrinsic parallelism of the SVM algorithm has been carried out and, after that, a study of the implementation space of this algorithm using a dataflow language called RVC-CAL has been performed. This language simplifies the parallelization of the system and simplifies the test of different system distributions. Specifically, this research is aimed at reducing the prediction time of the SVM classifier using RVC-CAL.

The results obtained show that the adaptation of the LibSVM code to the RVC-CAL language implies a reduction of the hyperspectral image classification time. Additionally, as this algorithm supports a pixel-wise processing strategy, the parallelization of the system is intuitive. In this line, to study the implementation space, two different networks where the pixel-wise parallel classification is exploited have been proposed and compared.

The study of the implementation space has shown that the whole processing chain of the SVM algorithm is completely parallelizable due to the pixel independency of the system. In consequence, using a maximum of 3 cores to process the image in parallel, average speedups of 2.85 and 2.50 have been obtained for Case 1 and Case 2, respectively.

Finally, the reduction of the parallelization process complexity has been demonstrated using the RVC-CAL language. As the SVM classifier supports a pixel-level parallelism, this algorithm has been a perfect candidate to test the capabilities and limitations of this dataflow language. RVC-CAL provides thus a powerful tool to extract the algorithm inherent parallelism.

ACKNOWLEDGEMENTS

The authors would like to thank the HELICoiD project (FP7-ICT-2013.9.2 (FET Open) 618080) for its financial support.

REFERENCES

[1] C.I. Chang, Hyperspectral imaging: techniques for spectral detection and classification, vol. 1. Springer Science & Business Media, University of Maryland, Baltimore, 2003.

[2] D. Manolakis and G. Shaw, "Detection algorithms for hyperspectral imaging applications," Signal Processing Magazine, IEEE, vol. 19, nº 1, pp. 29-43, January 2002.

[3] M. Govender, K. Chetty and H. Bulcock, "A review of hyperspectral remote sensing and its application in vegetation and water resource studies," Water Sa, vol. 33, nº 2, 2007.

[4] E. Adam, O. Mutanga and D. Rugege, "Multispectral and hyperspectral remote sensing for identification and mapping of wetland vegetation: a review," Wetlands Ecology and Management, vol. 18, nº 3, June 2010.

[5] E.K. Hege, D. O'Connell, W. Johnson, S. Basty and E.L. Dereniak, "Hyperspectral imaging for astronomy and space surviellance," in Optical Science and Technology, SPIE's 48th Annual Meeting, International Sociesty for Optics and Photonics, pp. 380-391, January 2004.

[6] M.E. Martin, et al, "Development of an advanced hyperspectral imaging (HIS) system with applications for cancer detection," Annals of biomedical engineering, vol. 34, nº 6, pp. 1061-1068.

[7] S. Kuching, "The performance of maximum likelihood, spectral angle mapper, neural network and decision tree classifiers in hyperspectral image analysis," Journal of Computer Science, vol. 3, nº 6, pp. 419-423, 2007.

[8] L. Ma, M.M. Crawford and J. Tian, "Local manifold learning-based-nearest-neighbor for hyperspectral image classification," in Geoscience and Remote Sensing, IEEE Transactions on, nº11, pp. 4099-1409, November 2010.

[9] N. Ma, S. Wang, S. Ali, X. Cui and Y. Peng, "High efficiency on-board hyperspectral image classification with Zynq SoC," in MATEC Web of Conferences, vol.45, p- 05001, EDP Sciences, 2016.

[10] HELICoiD (Hyperspectral Imaging Cancer Detection) project website (last access on 09/08/2016): http://www.helicoid.eu

[11] G. Camps-Valls and L. Bruzzone, "Kernel-based methods for hyperspectral image classification," Geoscience and Remote Sensing, IEEE Transactions on, vol. 43, nº 6, pp. 1351-1362, June 2005.

[12] E. Bezati, M. Mattavelli and M. Rauler, "Rvc-cal dataflow implementations of mpeg avc/h. 264 cabac decoding," IEEE Conference on Design and Architectures for Signal and Image Processing (DASIP), pp. 207-213, October, 2010.

[13] F. Melgani and L. Bruzzone, "Classification of hyperspectral remote sensing images with support vector machines," IEEE Transactions on geosciences and remote sensing, vol. 42, no. 8, pp. 1778-1790, August, 2004.

[14] Y. Bazi and F. Melgani, "Toward an optimal SVM classification system for hyperspectral remote sensing images," IEEE Transactions on geosciences and remote sensing, vol. 44, no. 11, pp. 3378-3385, November, 2006.

[15] S. Herrero-Lopez, J.R. Williams and A. Sanchez, "Parallel multiclass classification using SVMs on GPUs," Proceedings of the 3rd Workshop on general-purpose computation on graphics processing units, pp. 2-11. 2010.

[16] Z. Chen and J. Chua, "Implementing parallel SMO to train SVM on CUDA-Enabled systems," 2012.

[17] J. Eker and J. Janneck, "CAL language report: Specification of the CAL actor language," 2003.

[18] R. Wu et al, "Exploring the Concurrency of an MPEG RVC Decoder Based on Dataflow Program Analysis," IEEE Transactions on Circuits and Systems for Video Technology, vol. 19, no. 11, pp. 1646-1657, November, 2009.

[19] F. Melgani and L. Bruzzone, "Classification of hyperspectrral remote sensing images with support vector machines," Geoscience and Remote Sensing, IEEE Transactions on, vol. 42, nº 8, pp. 1778-1790, August 2004.

[20] T.F. Wu, C.J. Lin and R.C. Weng, "Probability estimates for multi-class classification by pairwise coupling," Journal of Machine Learning Research, vol. 5, pp. 975-1005, 2005.

[21] R. Salvador et al, "Demonstrator of the HELICoiD tool to detect in real time brain cancer," Design and Architectures for Signal and Image Processing (DASIP), 2016 Conference on, pp. 1-2, IEEE, October 2016.

[22] C.C. Chang and C.J. Lin, "LIBSVM: a library for support vector machines," in ACM Transactions on Intelligent Systems and Technology (TIST), vol. II, article 27, New York, 2011.