IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 20, 2019, accepted December 24, 2019, date of publication January 3, 2020, date of current version January 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2963939

Towards Real-Time Computing of Intraoperative Hyperspectral Imaging for Brain Cancer Detection Using Multi-GPU Platforms

GIORDANA FLORIMBI[1], HIMAR FABELO[2], EMANUELE TORTI[1], (Member, IEEE), SAMUEL ORTEGA[2], MARGARITA MARRERO-MARTIN[2], GUSTAVO M. CALLICO[2], (Member, IEEE), GIOVANNI DANESE[1], AND FRANCESCO LEPORATI[1]

[1]Department of Electrical, Computer and Biomedical Engineering, University of Pavia, 27100 Pavia, Italy
[2]Institute for Applied Microelectronics (IUMA), University of Las Palmas de Gran Canaria (ULPGC), 35017 Las Palmas de Gran Canaria, Spain

Corresponding author: Giordana Florimbi (giordana.florimbi01@universitadipavia.it)

This work was supported in part by the Canary Islands Government through the ACIISI (Canarian Agency for Research, Innovation and the Information Society), ITHaCA Project Hyperspectral Identification of Brain Tumors, under Grant ProID2017010164, in part by the Spanish Government through PLATINO Project under Grant TEC2017-86722-C4-4-R, in part by the European Commission through the FP7 Future and Emerging Technologies (FET) Open Programme under Grant ICT-2011.9.2, and in part by the European Project HELICoiD under Grant 618080.

ABSTRACT Several causes make brain cancer identification a challenging task for neurosurgeons during the surgical procedure. The surgeons' naked eye sometimes is not enough to accurately delineate the brain tumor location and extension due to its diffuse nature that infiltrates in the surrounding healthy tissue. For this reason, a support system that provides accurate cancer delimitation is essential in order to improve the surgery outcomes and hence the patient's quality of life. The brain cancer detection system developed as part of the "HypErspectraL Imaging Cancer Detection" (HELICoiD) European project meets this requirement exploiting a non-invasive technique suitable for medical diagnosis: the hyperspectral imaging (HSI). A crucial constraint that this system has to satisfy is providing a real-time response in order to not prolong the surgery. The large amount of data that characterizes the hyperspectral images, and the complex elaborations performed by the classification system make the High Performance Computing (HPC) systems essential to provide real-time processing. The most efficient implementation developed in this work, which exploits the Graphic Processing Unit (GPU) technology, is able to classify the biggest image of the database (worst case) in less than three seconds, largely satisfying the real-time constraint set to 1 minute for surgical procedures, becoming a potential solution to implement hyperspectral video processing in the near future.

INDEX TERMS Hyperspectral imaging, high performance computing, graphic processing unit, parallel processing, parallel architectures, image processing, biomedical engineering, medical diagnostic imaging, cancer detection, supervised classification, support vector machines, K-Nearest neighbors, principal component analysis, K-means, majority voting.

I. INTRODUCTION

Nowadays, the use of hyperspectral imaging (HSI) for cancer tissue analysis and identification is continuously increasing in the literature [1]–[3]. In particular, HSI is a promising powerful technique for the identification and delineation of the cancer area during surgical procedures, facilitating the accurate resection of the tumor tissue, being a non-invasive, non-contact and non-ionizing technique [1]. HSI is an imaging modality able to obtain both spatial and spectral information from the scene that is being captured, measuring the reflected, absorbed or emitted radiance at certain wavelengths (also called spectral channels) [4].

In contrast with the standard RGB cameras that only captures information in three spectral channels (red, green and blue) within the visual range (400-700 nm), hyperspectral

The associate editor coordinating the review of this manuscript and approving it for publication was Robert P. Schumaker.

(HS) cameras are able to capture a very large number of spectral channels within the visual range and beyond it. Normally, these cameras cover the VNIR (visual and near-infrared) range (400-1000 nm), the NIR (near-infrared) range (900-1700 nm) or the SWIR (short-wavelength infrared) range (1000-2500 nm), depending on the type of sensor employed for the acquisition of the data [5].

Due to the high dimensionality of the acquired data using HS cameras, and the necessity of the intraoperative surgical guidance tools to be executed in real-time, the use of highly parallel high-performance processing platforms to process the data becomes mandatory. Among the different available parallel technologies, the Graphical Processing Units (GPUs) are the most appealing solutions to execute intrinsically parallel algorithms that elaborate a huge amount of data. Several aspects make the GPU technology preferable compared to other High Performance Computing (HPC) solutions for this specific application. This kind of device is hosted inside a standard desktop case that can be placed inside the operating room, where the classification takes place. Moreover, it is worth noticing that the operating room does not present power restrictions, so it is possible to introduce a desktop system equipped with one or more GPUs to elaborate the images. It is also important to highlight that this technology is characterized by shorter development times and more flexibility compared to other devices, such as Field Programmable Gate Arrays (FPGA) or Application Specific Integrated Circuit (ASIC). In these last cases, it is also difficult to apply modifications or extensions to the original design, since they typically involve a complete re-design of the whole architecture. A crucial aspect to consider is that a desktop-based solution allows to locally process data in real-time, without resorting to remote host processing systems. This last choice should be avoided in order to not transfer sensitive data (such as brain images from patients) over general and public networks. Moreover, the transmission of HS images, whose size is in the order of hundreds of Megabytes, presents unpredictable delays, slowing down the surgery duration. These considerations, and the results that will be presented in this work, reveal that a GPU solution within a local desktop PC is currently highly suitable for HSI classification in the intraoperative brain tumor detection application.

GPUs have already been used to analyze HS images in other fields [6]–[8], obtaining huge speed-up compared to the serial versions of the algorithms. As far as the brain cancer detection is concerned, authors have already exploited these devices to accelerate supervised [9] and unsupervised [10] classification algorithms achieving real-time developments.

The main goal of the work presented in this manuscript is the use of a multi-GPU platform to accelerate the intraoperative HS brain cancer detection algorithm developed during the execution of the European HELICoiD project [11], [12] to provide real-time classification during neurosurgical procedures. Due to the importance of a precise resection of brain tumors [13]–[15] and the lack of accurate tools to assist in this task [16]–[20], HSI is becoming a promising imaging technique to develop a surgical aid visualization tool that will assist neurosurgeons during surgery. Once the HS image of the brain is acquired, the aim of the system is to assign to each pixel a label representing one of the following classes: *normal tissue*, *tumor tissue*, *hypervascularized tissue* and *background*. The surgeon can recognize the different tissues from the classification map provided by the system, where each class is represented by a specific color. A proof-of-concept of this surgical aid visualization tool was already implemented onto a system based on a combination between a CPU (Central Processing Unit) and a many-core platform, obtaining a processing time of $\sim$1 minute, achieving an average speed-up factor of 24$\times$ [21]. In this work, the aim is to achieve real-time execution developing a parallel version of the entire HS brain cancer detection algorithm onto a multi-GPU system. This manuscript is organized as follows. In Section II authors describe the acquisition system, the image database and the test systems used. Moreover, authors present the serial and parallel versions of the algorithms included in the brain cancer classification system. At the end, the graphical user interface is briefly described. In Section III the results are presented and discussed. Finally, in Section IV authors present the conclusion of this work.

II. MATERIALS AND METHODS

This section describes the database employed to test the proposed implementations as well as the acquisition system employed to obtain these data. In addition, the high performance processing platforms and the HS brain cancer detection algorithm used for the implementations are presented and explained. Finally, the different approaches to parallelize and implement the algorithm onto the processing platforms are described.

A. ACQUISITION SYSTEM AND HS IMAGE DATABASE

The developed GPU-based implementations were installed onto a customized intraoperative HS acquisition system equipped with a VNIR *pushbroom* camera (Hyperspec® VNIR A-Series, Headwall Photonics Inc., Fitchburg, MA, USA) [21]. The system is able to capture HS images covering the spectral range from 400 to 1000 nm, with a maximum spatial resolution of 1004 $\times$ 1787 pixels (129 $\times$ 230 mm) and 826 spectral bands. Since the HS camera is based on the pushbroom method, it provides high spectral resolution and relatively high spatial resolution. However, the sensor is only able to capture the complete spectral dimensions and one spatial dimension of the scene, being necessary a spatial scanning to obtain the complete HS cube. By shifting the camera's field of view with respect to the scene, the second spatial dimension is acquired. The system includes also an illumination device capable of emitting cold light in the range between 400 to 2200 nm. A Quartz Tungsten Halogen (QTH) lamp is connected to the cold light emitter through a fiber optic cable to avoid the high temperatures of the light in the exposed brain surface. Using this acquisition system, five HS images

were obtained intraoperatively at the University Hospital Doctor Negrin of Las Palmas de Gran Canaria (Spain) from four different adult patients that underwent craniotomy for resection of intra-axial brain tumor. The patients had a confirmed grade IV glioblastoma tumor by histopathology. The study protocol and consent procedures were approved by the *Comité Ético de Investigación Clínica-Comité de Ética en la Investigación* (CEIC/CEI) of University Hospital Doctor Negrin and written informed consent was obtained from all subjects.

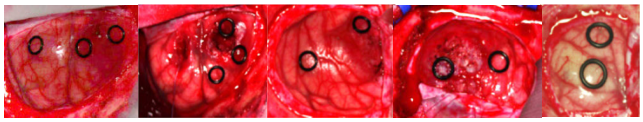

FIGURE 1. Synthetic RGB representations of the HS cubes from (a) P1C1, (b) P1C2, (c) P2C1, (d) P3C1, (e) P4C1.

Fig. 1 shows the synthetic RGB representation of each HS cube of the HS brain cancer image database employed in this work, while Table 1 details the characteristics of each HS cube, where PXCY stands for Patient X and Capture Y. These HS cubes can be downloaded in the supplementary material of this manuscript and the entire database obtained during the HELICoiD project execution was published in [11] (open access). The images in Fig. 1 are characterized by the presence of black rubber ring markers employed for the pathological assessment of the image labeling. The sterilized markers were placed by the surgeon in some specific areas, where he/she was confident to belong to normal and/or tumor tissue. In this last case, the tumor presence was confirmed by a biopsy, followed by a histopathological analysis. During the classification, the markers are assigned to the *background* class. Further details about the protocol used for image acquisition can be found on [12].

TABLE 1. Specifications of the HS image database.

Image ID	#Pixels	Dimensions [width x height x bands]	Size [MB]
P1C1	251,532	548 x 459 x 826	407.41
P1C2	264,408	552 x 479 x 826	428.23
P2C1	219,232	496 x 442 x 826	355.19
P3C1	184,875	493 x 375 x 826	299.65
P4C1	124,033	329 x 377 x 826	201.24

B. TESTS SYSTEMS

HSI classification during neurosurgical operations strongly requires a real-time elaboration. This constraint can be satisfied exploiting HPC systems able to elaborate large amount of data as fast as possible. Since HS images are characterized by a high number of information, the hybrid systems considered in this work are equipped with some of the most powerful NVIDIA GPUs currently available in the market, characterized by more than a thousand of processing cores and large memories. Table 2 presents the characteristics of the three systems used in these tests.

TABLE 2. CPU and GPU characteristics of the different testing systems.

System ID		TS1	TS2	TS3
CPU	Model	Intel i7 3770	Intel i7 6700	Intel i9 9900X
	Clock Rate [GHz]	3.40	3.40	3.50
	RAM [GB]	8	32	128
GPU	Model	TESLA K40	GTX 1060	RTX 2080
	#Cores	2880	1152	2944
	#SMXs	15	9	46
	Clock Rate [GHz]	0.875	1.78	1.71
	RAM [GB]	12	3	8
	Architecture	Kepler	Pascal	Turing
	#Boards	2	1	2

The GPUs choice has been done aiming at covering a wide range of manycore architectures, evaluating one specific board for scientific computations and other two general consumer GPUs. The Test System (TS) 1 is equipped with two NVIDIA Tesla K40 GPU, based on the NVIDIA Kepler architecture (compute capability 3.5). This board is specific for scientific computation and therefore it does not present a graphical output. These GPUs are connected to an Intel i7 3770 CPU. On the other hand, the consumer GPUs have been included in TS2 and TS3. The former is equipped with an NVIDIA GTX1060 GPU which is based on the Pascal architecture (compute capability 6.0). In this case the board is connected to an Intel i7 6700. Finally, TS3 is equipped with two NVIDIA RTX2080 GPU, based on the Turing architecture (compute capability 7.5), connected to an Intel i9 9900X CPU. All the boards are connected to their CPU through a PCI Express 3.0 bus.

C. BRAIN CANCER DETECTION SYSTEM

The classification system (CS) developed in the HELICoiD project performs a hybrid spatial/spectral and supervised/unsupervised classification of the HS images acquired during the surgery. Fig. 2.A shows an overview of the classification algorithm [12].

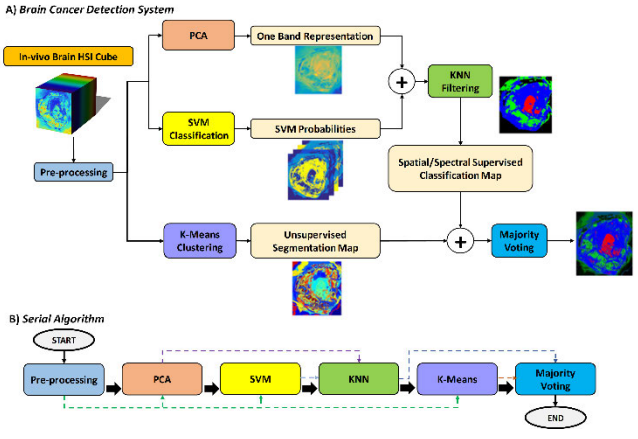

FIGURE 2. A) Algorithms constituting the brain cancer detection system. The HSI cube is the input of the system. It is pre-processed and sent to the supervised (PCA, SVM, and KNN) and unsupervised (K-means) classification branches of the system. The results of these two processing flows are combined with the Majority Voting algorithm. B) Serial algorithm of the classification system. Black arrows indicate the algorithm flow and the colored dashed arrows represent data dependencies.

At first, the image is stored as a *raw* file that is pre-processed in order to homogenize the spectral signature of each pixel. The pre-processed image is the input of the two main parts of the CS: one performs a spatial-spectral supervised classification while the other accomplishes an unsupervised clustering segmentation. Concerning the supervised classification, the Principal Component Analysis (PCA) algorithm computes a one-band representation of the HS cube, while the Support Vector Machine (SVM) classifier performs a pixel-wise supervised classification and generates a 4-class probability map. The K-Nearest Neighbors (KNN) algorithm is used as a filter to integrate the PCA and the SVM outputs in order to improve the results of the spectral classification by adding spatial domain information [22]. On the other hand, the unsupervised stage of the CS is based on the K-means clustering algorithm that creates a segmentation map where the boundaries of the different spectral regions are well delineated. Nevertheless, the segmentation performed by the unsupervised classifier cannot be directly related with the different tissues or materials that are present in the image. Finally, the Majority Voting algorithm combines these two main parts of the CS whose output is the result of the hybrid classification. Fig. 2.B depicts the serial algorithm of the CS. The image shows both the flow (black arrows) and the data dependencies (colored dashed arrows) among the different steps.

The next paragraphs will provide a more detailed description of each step of the CS, presenting both the serial and parallel versions of each algorithm. As stated before, a crucial specification that the system has to satisfy is the real-time elaboration in order to not slow-down the surgical operation. In this particular case, the real-time constraint is defined by a limit of 1 minute, which is the time that the camera takes to capture the HS image due to the pushbroom capturing technique. Firstly, a parallel version of each algorithm was developed, trying to accelerate their highest computational parts. At the end, these parallel codes have been merged in order to get the most efficient version of the complete CS. It should be pointed out that merging the parallel algorithms is not a simple system integration since it implies some research challenges. In fact, a simple pipelining of the parallel versions of the algorithms cannot ensure a real time classification of a hyperspectral image. The main goal of this work is to provide a complete parallel system, which efficiently integrates all the parallel algorithms. To this aim, it is of critical importance to entirely develop each algorithm keeping in mind that it will be part of a more complex system.

1) PRE-PROCESSING CHAIN

The captured HS image could present significant signal variations due to a non-uniform illumination within the brain surface. Thus, the pre-processing algorithm aims to correct these differences and to homogenize the spectral signatures of the *in-vivo* HS cubes [23]. The pre-processing chain is based on three main steps: image calibration, band and noise reduction and data normalization.

To correct the signal variations of the input data, in the calibration phase, *white* and *dark reference* images are acquired in the same operating theatre and with the same illumination conditions where the HS cube will be captured. The white image is obtained by using a white standard reference tile while the black one is obtained by keeping the shutter camera closed. The calibrated image (CI in (1)) is computed as follows:

$$CI = 100 \times \frac{RI - DR}{WR - DR} \tag{1}$$

where RI is the input raw image and, WI and DI are the white and dark reference images [23].

Once the HS image is calibrated, a band reduction is performed in order to remove the extreme noisy bands that do not provide useful information due to the sensor underperformance. Moreover, it has been observed that consecutive bands provide redundant information, due to the extremely high spectral resolution. For this reason, the reflectance values of seven contiguous bands are averaged every five bands: the reduced HS cube is characterized by 128 bands [9]. This reduction allows saving computational time and memory due to the diminished number of data to elaborate.

The last phase of the pre-processing chain is the data normalization needed to avoid different extreme radiation intensity levels between the pixels captured at different heights. This fact could produce a classification based on the pixel brightness without actually taking into account its spectral signature. For this reason, the pixel normalization is performed to obtain the spectral signature amplitude of each pixel between 0 and 1.

The flow of the pre-processing algorithm parallel code (Fig. 3.B) presents three steps. Starting on the host, where the input image and the white and dark images are read and stored, data are then transferred to the device where the pre-processing chain can start with the calibration (*1st phase* in Fig. 3.B). It is important to highlight that the data transfer from host to device is managed using the CUDA *streams* [24], which allow to overlap the data transfers and the operations performed on the device. In this case, a suitable number of streams is created. Each stream manages the transfer of a row (i.e. the current frame acquired by the camera) and the computation of the calibrated image is performed exploiting the cuBLAS library (NVIDIA) which includes highly optimized linear algebra routines [25]. Once the row transfer is completed, the cuBLAS function can start the computation without waiting for the transfer of the following row. In this way, the computational time is reduced compared to a serial elaboration. Once the image is calibrated, the following two phases can start. A custom kernel performs the normalization while the band reduction is done sending to this kernel only the final bands that have to be considered. In this way, these two phases (*2nd and 3rd phases* in Fig. 3.B) are performed concurrently.

2) PRINCIPAL COMPONENT ANALYSIS (PCA)

Despite being characterized by a large amount of data, even after the reduction obtained with the previous step, the HS

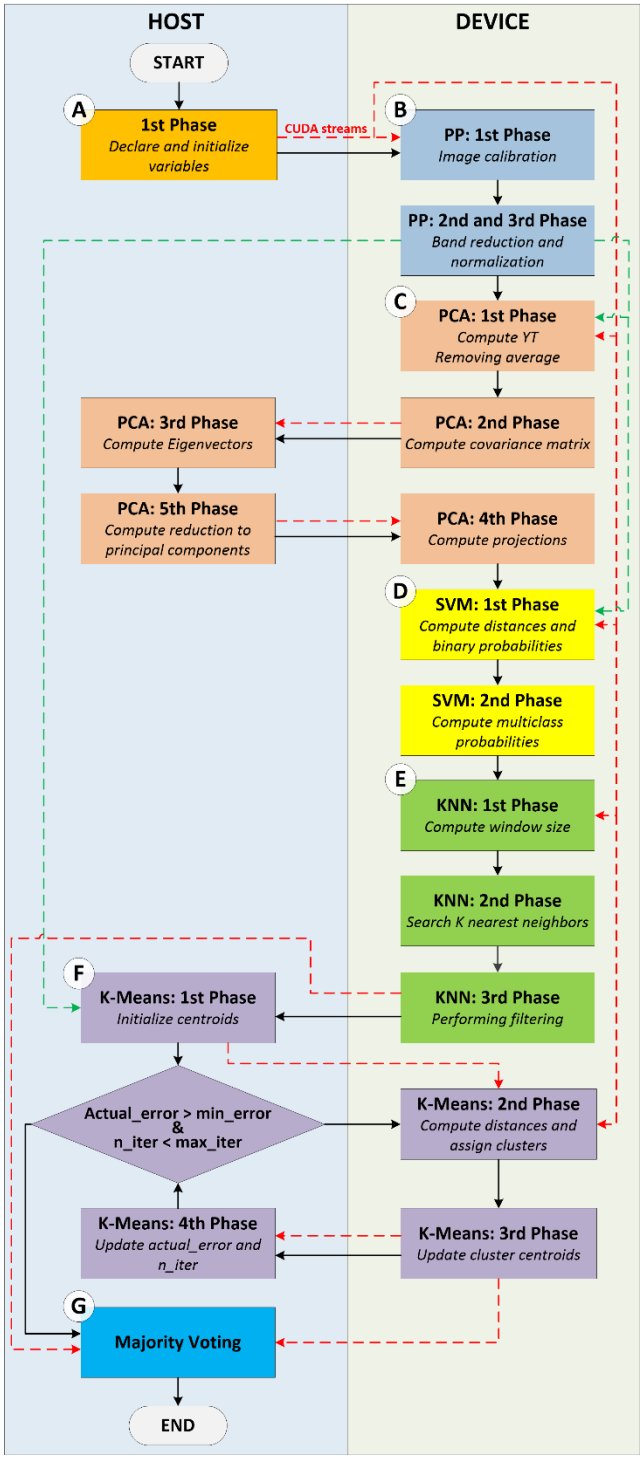

FIGURE 3. Flow diagram of the complete system parallel code on single device (CS-Single). The black arrows indicate the flow of the algorithm, the green ones represent the pre-processed image sent as input to the different algorithms. The red arrows indicate the data transfers. ***CUDA streams*** indicates that the transfer from host to device is performed exploiting streams to overlap data transfer and computation. A) Variables declaration and initialization. B) Pre-processing. C) PCA dimensional reduction. D) SVM classification. E) KNN filtering. F) K-Means segmentation. G) Majority Voting.

image dimensionality can be decreased even more through the PCA algorithm. This technique converts the original data into a new subspace, selecting and accumulating the most important information in the first bands [26]. PCA reduces those data by computing the covariance matrix of the HS cube and its eigenvector decomposition. Then the image is projected into the sub-space described by these eigenvectors and, at the end, the principal components or bands are selected.

The serial implementation of the PCA algorithm is organized in sequential steps [26]: after the variables declaration and initialization, the algorithm computes the transpose of the input matrix $Y_{N\times M}$, where N is the number of pixels and M is the number of bands. This step continues computing the average of the elements present in each row of Y^T. Then, each average is removed from each element of the corresponding band, i.e. the row of the matrix Y^T, obtaining a new X matrix. The covariance matrix $C_{M\times M}$ is then computed as the product between the matrix $X_{M\times N}$, output of the previous step, and its transpose. In the next step, the eigenvectors E, associated with the covariance matrix C, are extracted exploiting the Jacobi method, a fast algorithm capable of extracting the eigenvectors while computing the input matrix eigenvalues [26]. Then, the algorithm performs the projection of the input image onto the set of the extracted eigenvectors. Finally, the first P principal components are selected in order to obtain a matrix with reduced dimension $N\times P$. In this work, the output consists in an array whose dimension is $N\times 1$, since only one principal component is selected.

The serial code profiling showed that the most consuming part of the algorithm is the covariance matrix computation, which takes more than the 70% of the serial execution processing time. In order to reduce the computational time of the algorithm, two parallel versions have been developed. In these codes, not only the covariance matrix computation is performed on the GPU. In order to accelerate all the stages of the algorithm, the whole computation is carried out by the device. The difference between the two versions (PCA1 and PCA2) is in the part where the eigenvector computation is elaborated.

In fact, in PCA1, this step has been computed exploiting a suitable function of the cuSOLVER library [27] (developed by NVIDIA), capable of reproducing the Jacobi method. After a code profiling it has been noticed that the eigenvectors computation takes longer on the GPU than on the host. For this reason, PCA2 has been developed and the eigenvector computation has been moved back on the host.

Both versions start with the GPU memory initialization, included in the 1^{st} *phase* of Fig. 3.A. The pre-processed image, already stored in the GPU global memory, is taken as input from the pre-processing (green dashed line in Fig. 3). The transpose of the input matrix is computed (Y^T) exploiting the *cublasSgeam* routine, belonging to the cuBLAS library. Then, two main operations are performed: the average computation of each row of the matrix Y^T and the subtraction of these averages from each element of the corresponding rows (1^{st} *phase* in Fig. 3.C). The first task is evaluated exploiting the highly optimized function *cublasDasum* that belongs to the cuBLAS library [25]. In this case, a number of streams equal to the number of bands (i.e. the number of rows of the

matrix) is created. Each stream manages the mean computation of its elements through the *cublasDasum* function. After that, all the averages are computed and subtracted from each element of the related row. This computation is performed by a custom kernel. Furthermore, in this case, each stream manages the call of the kernel which removes the average from each element simultaneously. This kernel is characterized by a *grid* whose dimension is computed as the ratio between the number of pixels and the number of threads in a block (i.e. 32 in this case). The output of the PCA *1st phase* is the matrix $X_{M\times N}$. X represents the *2nd phase* input where the covariance matrix C is computed. This step exploits another function of the cuBLAS library, *cublasDgemm*, which allows computing the product between two matrices. The end of the *2nd phase* is characterized by another custom kernel which simultaneously divides each element of the covariance matrix C by the number of pixels. In the case of the PCA1 version, the code starts to compute the eigenvectors on the device using the *syevj* routine of the cuSOLVER library. In order to use this routine, some variables and arrays have to be declared and initialized. These variables, together with the covariance matrix, are the inputs of the function which returns the sorted eigenvectors matrix (phase not shown in Fig. 3.C). On the other hand, PCA2 computes the eigenvectors on the CPU and, for this reason, the covariance matrix is transferred from the device to start the eigenvectors computation on the host (*3rd phase*). Moreover, in PCA2, after this step, the *5th phase* is anticipated compared to the serial version in order to select the principal components P on the host. In this way, only the eigenvectors related to these principal components are copied from the host to the device in order to reduce the transfer time and the memory occupancy. At this point, in both versions, the projections are computed as described before exploiting the cuBLAS functions (*4th phase* in Fig. 3.C). Finally, the PCA result (in our case a one-band representation of the HS cube) is stored on the GPU global memory.

Table 3 shows the elaboration times of the serial PCA algorithm and the two parallel versions (PCA1 and PCA2), together with the related speed-up (between brackets). In particular, for each parallel code, the table provides the time related to the elaboration of the whole code (“*Parallel*” column) and the one related only to the steps of the algorithm (“*Steps*” column). In this last case, not all the phases present both in PCA1 and in PCA2 are considered in the measure of the time. In fact, the time related to the CUDA context creation, to the first and last transfers and to the initialization of the variables is not taken into account since it is the same in both versions. In this specific analysis, the aim is to select the fastest PCA code to be included in the complete system. For this reason, it is important to focus on the time of the stages since the steps of the CUDA context creation or the initialization will be executed once for the entire complete system. As stated before, the only difference between the stages of the two PCA versions is the eigenvectors computation, managed by the device in the PCA1 case and by the host in PCA2. Table 3 shows that both parallel versions

TABLE 3. Elaboration times of the serial, pca1 and pca2 versions exploiting the three test systems.

TS1					
		PCA1		PCA2	
Image ID	Serial [s]	Parallel [s]	Steps [s]	Parallel [s]	Steps [s]
P1C1	2.227	1.017 (2.19×)	0.191	0.832 (2.68×)	0.076
P1C2	2.577	1.024 (2.52×)	0.208	0.919 (2.80×)	0.084
P2C1	1.937	0.956 (2.03×)	0.183	0.732 (2.65×)	0.070
P3C1	1.623	0.917 (1.77×)	0.182	0.629 (2.58×)	0.060
P4C1	1.082	0.800 (1.35×)	0.181	0.626 (1.73×)	0.043
TS2					
		PCA1		PCA2	
Image ID	Serial [s]	Parallel [s]	Steps [s]	Parallel [s]	Steps [s]
P1C1	2.696	1.516 (1.78×)	0.369	1.217 (2.22×)	0.116
P1C2	2.889	1.545 (1,87×)	0.380	1.235 (2.34×)	0.124
P2C1	2.418	1.400 (1.73×)	0.362	1.103 (2.19×)	0.104
P3C1	2.114	1.340 (1.58×)	0.351	1.030 (2.05×)	0.089
P4C1	1.372	1.214 (1.13×)	0.319	0.910 (1.51×)	0.064
TS3					
		PCA1		PCA2	
Image ID	Serial [s]	Parallel [s]	Steps [s]	Parallel [s]	Steps [s]
P1C1	2.345	2.056 (1.14×)	0.552	1.487 (1.58×)	0.058
P1C2	2.506	2.063 (1.21×)	0.560	1.593 (1.57×)	0.062
P2C1	2.045	2.004 (1.02×)	0.546	1.421 (1.44×)	0.051
P3C1	1.720	1.697 (1.01×)	0.544	1.348 (1,28×)	0.045
P4C1	1.675	1.422 (1.18×)	0.527	1.273 (1.32×)	0.032

achieve a speed-up compared to the serial implementation. Moreover, it is possible to highlight that the PCA2 code is the fastest, considering all the test systems. Focusing on the “*Steps*” column, which is the different part of the two implementations, it is possible to notice that the eigenvectors computation on the host is more efficient than the use of the cuSOLVER routine on the device.

In ths specific case, the data transfers and the computation on the host are faster than the cuSOLVER context creation and computation on the device. Despite the increased number of transfers, the PCA2 version resulted to be the fastest one and, for this reason, it is the one included in the complete system parallel codes and whose flow is shown in Fig. 3.C.

3) SUPPORT VECTOR MACHINES (SVM_S)

The SVM classifier is a machine learning technique that computes the probability of a pixel to belong to each different

class under study. The pixel will be assigned to the class with the highest probability. In our case, the SVM classifier output is not the label that represents the class but a probability map that contains the likelihood of each pixel to belong to each one of the four defined classes. This output will be one of the inputs of the KNN algorithm, which is the following step. The SVM classifier was selected for the brain cancer detection system because it provides good performance for HS data and in the present experiment conditions: limited training dataset and real-time constraint [12], [28].

The SVM algorithm is composed of two main parts: training and prediction. Since the SVM performs a supervised classification, a labeled dataset is needed in order to train the model which will be used in the prediction phase. This dataset was obtained following the procedure explained in [12]. The training phase of the SVM classifier was developed using MATLAB®, exploiting the LIBSVM library [29]. It is not included in this work since it is an *offline* step of the system: the idea is to previously generate the supervised model and, then, send it as an input of the prediction part of the algorithm. The SVM algorithm faces a multiclass problem with a *one-against-one* or binary strategy: each pair of classes is evaluated in order to find the best hyperplane that separates their elements in the multidimensional space. The number of binary evaluations, and so the number of the computed linear hyperplanes, depends on the number of classes considered in the classification.

The SVM prediction algorithm inputs are the samples to be classified, a set of support vectors, the bias, and the sigmoid function parameters. After all these variables are read and stored, the algorithm elaborates two steps for each pixel. The former firstly computes the distance between the sample (i.e., the pixel) and the considered hyperplane, and then estimates the binary probability of the pixel to belong to the two classes under study. The latter computes the multiclass probabilities starting from the binary ones elaborated in the previous step. In particular, a *for* loop iteratively refines the values of the probabilities of a pixel associated with a class. The value of each probability is incrementally modified as long as the difference with the value of the previous iteration is under a certain threshold or if the maximum error is reached. When one of these two cases is verified, the multiclass probabilities of the sample are computed. Both the threshold and the error can be set by the user and, for this reason, they are provided as input to the algorithm.

The SVM output is a probability map containing the multiclass probabilities of all the samples, which are the pixels of the HS image. It is important to underline that the probability computation is evaluated independently for each sample: this will be an essential aspect in developing a fast pixel-wise classification. Even if the analysis conducted on the serial SVM results proves that it is a highly optimized algorithm, since its computational time is very low, a parallel version of this algorithm has been developed to alleviate the host load and reduce data transactions. In the parallel implementation, the classification of multiple pixels can be computed simultaneously. The reason is that a parallel version of this classifier could be very useful in the development of the complete system by avoiding some data transfers. In fact, if the complete system will perform the PCA and the KNN on the device, performing even the SVM on GPU could be more efficient thanks to the fewer I/O transfers from the host to the device and vice versa. The fact that the multiclass probability computation is independent for each sample eases the development of the SVM parallel version.

The flow of the SVM parallel code is shown in Fig. 3.D. The inputs acquisition and the variables declaration and initialization are performed on the host (Fig. 3.A). In particular, all the outputs of the SVM training phase are stored in a linear matrix in order to perform a fast and efficient transfer (red dashed line from *1st phase* (A) to SVM 1st *phase* (D) in Fig. 3). Moreover, the other SVM input, i.e. the pre-processed image, is already stored in the GPU global memory.

In this parallel version, the 1st *phase* of the algorithm is split in two steps: the former computes the distance between the sample and the hyperplane, and the latter elaborates the binary probabilities. The computation of the distance between the sample and the hyperplane is performed by a *cublasSgemm* function of the cuBLAS library. Then, the last part of the *1st phase* and the *2nd phase* can be pixel-wise parallelized: it is possible to create a custom kernel that simultaneously computes the binary and the multiclass probabilities for each pixel. Also in this case, the kernel is characterized by a *grid* whose dimensions are given by the ratio between the number of pixels and the number of threads. Each thread computes the multiclass probability of a single pixel. When this probability is updated, its final value is stored in a probability map on the GPU global memory.

4) K-NEAREST NEIGHBORS (KNN) FILTER

Recent uses of the KNN algorithm show that it is not restricted to a classification role, but it can also be used as a filtering technique [22], [30]. In the present work, this algorithm improves the spectral classification results by adding spatial information. In this scope, the KNN filter receives as input the probability map generated by the SVM classifier and the one-band representation of the HS image generated using the PCA algorithm. Considering each pixel of the HS image, the KNN performs a nearest neighbor searching step and a filtering step.

Concerning the first part, the nearest neighbors of a pixel are searched in a feature space that contains the pixel values and the spatial coordinates (2). In this equation, $I(q)$ is the normalized pixel value of the one-band representation and $l(q)$ and $h(q)$ refer to the normalized coordinates of pixel q. The parameter λ controls the balance between the pixel value and the spatial coordinates. For example, the spatial information is not considered when λ is equal to zero. Otherwise, if this value is higher than zero, more influence is given to the local neighborhood [30].

$$F(q) = (I(q), \lambda \cdot l(q), \lambda \cdot h(q)) \tag{2}$$

The process of finding the K nearest neighbors for each pixel involves the computation of its distance from the other pixels of the image. Authors in [22] have evaluated different distance metrics, finding that the Euclidean distance is the most suitable for this application; therefore, this metric is adopted in this work. For each pixel, the K nearest neighbors are selected after the distances have been sorted. Once the neighbors' selection is over, the filtering step can start. Taking into account the probability map Pc obtained by the SVM classifier, a set of new optimized probabilities for each pixel is computed as shown in (3).

$$O(q) = \frac{\sum Pc(s)}{K}, \quad s \in w_q \tag{3}$$

For each pixel, the algorithm computes a number of probabilities O equal to the number of SVM classes, which are four in this work. In (3), w_q indicates the nearest neighbors of the pixel q, s is the index related to each neighbor and K is the number of neighbors searched for each pixel [22]. The last step consists of assigning a label to each q corresponding to the highest value among the four optimized probabilities O computed for each pixel. In this way, a new classification map is computed.

It is important to highlight that the K nearest neighbors' selection is the most consuming part of the code from a computational point of view. For this reason, in [22] an optimization is introduced in order to save memory and execution time. Instead of searching the neighbors within the entire image, the selection is done inside a search window, i.e. a region that surrounds the pixel. The use of a smaller searching area is possible due to the values that authors in [12] assumed for the variables λ and K, which confer more importance to the pixel local neighborhood. For this reason, the probability to find neighbors near the pixel is higher than the probability to find smaller distances in further zones of the image. Several analyses were performed looking for a window dimension that produces the same (or almost the same) results as searching the neighbors in the whole image. The window, selected as a reference, surrounds the pixel in a symmetric way and it is characterized by a number of rows (*WSize*) of the image equal to 14 and by a number of pixels equal to *14 rows* $\times$ *total number of columns*. Authors in [22] have also evaluated different windows sizes in order to find a good compromise between the number of distances computed for each pixel and the final KNN filtering accuracy. In addition, the authors have introduced a GPU version of the KNN algorithm whose flow is included in Fig. 3.E.

The basic idea followed in the development of the parallel version is that each CUDA core has to assign a label to each pixel simultaneously. The declaration and initialization of all the variables are performed on the host. The first step of the KNN filtering algorithm on the GPU device concerns the execution of a kernel that evaluates the borders and the size of the windows in parallel through the pixels (*1st phase* in Fig. 3.E). In fact, in the parallel version is very important to define the window size for each pixel before the distance computation and the filtering. In this way each thread can copy, from the global to the shared memory of the GPU, only the PCA and SVM outputs that are required in the computations. Then, the results are copied to the global memory only at the end of the kernel execution. This step is crucial to decrease the execution time since the accesses to the global memory are very slow. In the *2nd phase* and in the *3rd phase* each thread evaluates in parallel the K nearest neighbors and the updated probability of a pixel, respectively [22]. The kernel of the *2nd phase* needs the PCA one-band representation to compute the distances between the pixels. Once this step is concluded, the filtering kernel (*3rd phase*) starts computing the optimized probability of each pixel for each class, taking into account the SVM probability map. Once the KNN algorithm execution ends on the GPU device, an array containing the labels of all the pixels is transferred to the host to be used in the Majority Voting algorithm, together with the K-means output.

5) K-MEANS CLUSTERING

The K-means algorithm performs an unsupervised classification since a labeled dataset is not needed. The algorithm generates groups, called *clusters*, separating tissues and materials present in the HS image, on the base of their spectral similarity. Each cluster centroid represents the spectrum of a particular material present in the image. The serial and parallel versions of this algorithm have been already presented in [10].

The serial algorithm starts with the random initialization of the K_C centroids, where the number of centroids K_C is given as input. Then, the algorithm iteratively evaluates the distances of the pixels with each centroid and assigns that pixel to the cluster with the lowest distance. Next, the centroids are updated. These two steps are repeated until one of the two stopping criteria is satisfied. The former is about an error threshold, the latter is related to a maximum number of iterations: both these data are given as inputs. The suitable values for these inputs have been experimentally found in [10]. The output is a segmentation map showing different clusters characterized by several colors but without any explicit meaning. In this system, the K-means is used to delineate the boundaries of the different spectral areas present in the HS image. Due to the high number of clusters used (25) and the richness of spectral information, the boundaries of the tissues are better delineated than when using the SVM with only four classes.

The parallel flow, shown in Fig. 3.F, starts on the host where the *1st phase* is performed and the centroids initialization is transferred to the device. Authors choose to initialize the centroids on the host because it is more efficient. In fact, the number of centroids to initialize is low (in this work it is 25); moreover, it is important to ensure that all the centroids are initialized at different values. Performing these operations on the device is not convenient because it would be necessary to use the cuRAND library and to check with a serial kernel that the generated values are different. This will affect the algorithm performance since the library

initialization takes time and the serial check of the initial values is not efficient on the device. A *while*-loop condition, evaluating the error and the number of iterations, is performed by the CPU. If the condition is true, the flow continues on the device where each thread of the kernel computes the distances between the pixel and all the centroids (2^{nd} *phase*). Moreover, each thread performs the cluster assignment for the pixel. The cluster centroids update, in the 3^{rd} *phase*, is computed by a simple kernel where the *i*-th thread manages the *i*-th centroid update. Once the update is completed, the variation between the previous and the actual centroids is evaluated exploiting the *cublasSasum* routine (cuBLAS library). This value is transferred to the host (red dashed arrow) to compute the error and to increment the iterations number (4^{th} *phase*). When the stopping criterion is satisfied, the flow proceeds with the last step of the classification system.

6) MAJORITY VOTING

The brain detection system presented in this work performs a hybrid classification, exploiting both the supervised and unsupervised learning outcomes. The first one provides a classification map in which each pixel belongs to a class: this map is obtained by exploiting the diagnosis information provided by the neurosurgeons and pathologists. Even if this kind of output assigns a label to each pixel, it does not provide an accurate delineation of the tumor area. On the other hand, the unsupervised classification provides a good association of similar areas in the image, even if the clusters cannot be directly related to the materials or types of tissue that are present in the image. These two approaches are computed independently but their results have to be merged in order to exploit their advantages and to obtain a final classification map. The Majority Voting is the approach selected to join these outputs, already used by authors of [13] and [31]. The two maps are merged to obtain the final hybrid result which shows each pixel belonging to a specific group, assigned on the base of the class of the supervised classification, also taking into account the clusters obtained by the unsupervised classification map [12].

D. PARALLEL VERSIONS OF THE COMPLETE SYSTEM

In the previous paragraphs, the parallel versions of all the algorithms that are part of the brain cancer detection system have been presented. The aim of this section is to explain how these algorithms have been merged exploiting the single-GPU and the multi-GPU paradigms in order to obtain the most efficient version of the system. The goal is to reach a real-time classification that, in this particular case, is defined by a limit of 1 minute. It is worth noticing that the development of the complete system parallel versions is not a simple algorithms integration. In fact, some issues have to be considered. A crucial role is covered by the memory management: firstly, memory allocations and transfers take into account the data path of the whole system, exploiting data dependencies in order to avoid multiple or redundant transfers. Moreover, a well-balanced computational load is mandatory, considering the system integration in the multi-GPUs version. The final parallel version represents the best configuration of memory allocations, data transfers and computational load of each algorithm for this particular application.

The serial flow of the complete system sequentially elaborates all the serial algorithms, as presented in Fig. 2.B. As it will be discussed in the Section III, this serial version of the complete classification system does not satisfy the real-time requirement. For this reason, the use of GPU technology is required to reduce the classification computational time. Three parallel versions have been developed: the first one exploits a single-GPU system, while the other two versions run on a multi-GPU system.

1) SINGLE-GPU PARALLEL VERSION (CS-SINGLE)

The first parallel version (*CS–Single*) of the brain cancer detection system is characterized by the execution of all the algorithms in cascade on the device (Fig. 3). The processing flow starts on the host where the input HS image and the other variables are declared and initialized. The user can set some input parameters in the graphical user interface (GUI), such as the number of neighbors K, the metric used in the distance computation, the *WSize* that concerns the KNN algorithm, the maximum number of iterations, the error threshold and the number of clusters K_C that concerns the K-means algorithm. The GUI (described in detail in Paragraph E) provides some default value of these variables, but the user can modify them. The HS image and all these variables are read and stored in the CPU memory and transferred on the device for the computation.

The other type of inputs is independent from the user: in fact, the *white* and *dark* calibration matrices and the variables belonging to the SVM model generated by the SVM training phase are also read, stored and transferred to the device. Once the data transfer is completed, the GPU parameters and variables are declared. For example, in this step, the *handle* used in the cuBLAS routines and the *streams* used in the pre-processing are created. Furthermore, the *grids* and *blocks* dimensions of the kernels are defined. Once the GPU initialization is completed, the pre-processing starts.

As explained in the Pre-processing paragraph (Section II.C), the *white* and *dark* matrices and the input image are transferred exploiting CUDA *streams*. Once the HS image is pre-processed, it is stored in the GPU memory and it is transferred to the host since it is used for the clusters initialization (K-means algorithm) performed on the CPU (green dashed lines in Fig. 3). The processing flow continues with the PCA and SVM algorithms whose input is the pre-processed HS image stored in the GPU global memory. Their outputs (the one-band representation of the HS image and the 4-class probability map) are stored in the GPU global memory to be used in the 2^{nd} and 3^{rd} KNN phases respectively.

The flow continues with the K-means algorithm, performed both on the host and on the device. Finally, the KNN and the K-means outputs are transferred to the

host for the Majority Voting computation. The RGB hybrid classification map, i.e. the system output, is shown to the user through the GUI. As it will be presented in Section III, this *single-GPU* version of the complete system classifies the hyperspectral image, even in the worst case, in less than half a minute, satisfying the targeted real-time constraint. To further reduce the computational time obtained with the *single-GPU* code, two different *multi-GPU* versions have been developed.

2) MULTI-GPU PARALLEL VERSION 1 (CS-MULTI1)

Analyzing the flows of the codes presented above, there are parts that could be executed simultaneously. For this reason, in this implementation, the idea is to execute these parts on different GPUs that work in parallel. In order to manage two boards in parallel, two sections (generated with the OpenMP API [32]) are created. Each one is characterized by a CPU-thread that manages the memory initialization, the variables declaration, the kernels launch and the data transfers of each board.

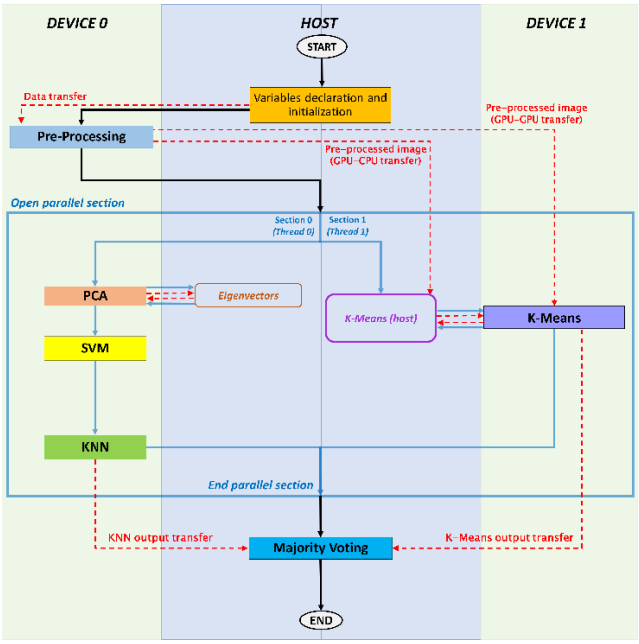

FIGURE 4. Flow diagram of the parallel code of the multi-GPU parallel version 1 (*CS-Multi1*) of the complete HS brain cancer detection algorithm.

The workflow of the first CS multi-GPU parallel version (*CS-Multi1*) is shown in Fig. 4. As in the *CS-Single* version, the variables declaration and initialization are performed on the host. The inputs are transferred to one of the two boards (in this case the *Device 0*) where the pre-processing is performed. The pre-processed HS image is stored in the *Device 0* global memory and it is also transferred both to the host, for the clusters initialization in the K-means, and to the other device (*Device 1*), for the K-means computation on GPU (red dashed lines). This transfer between devices is possible thanks to the GPUDirect technology [33], which allows devices to read and write CUDA host and device memory, avoiding unnecessary memory copies. In order to transfer data from the memory of one device to another, the first step consists in enabling the current GPU (in this case the *Device 0*) to access addresses of the *Device 1*. The function that allows this task is the *cudaDeviceEnablePeerAccess*. Then, the copy of the pre-processed image is performed by the function *cudaMemcpyPeerAsync*. Once the pre-processed image is transferred to the host and to the other device, a parallel section is opened exploiting the OpenMP statements (light blue box in Fig. 4), as shown in Algorithm A.

Algorithm A Parallel Sections Opening Pseudo Code

```
#pragma omp parallel sections
{
  #pragma omp section
  {
      In this section the thread 0 manages the Device
      0 and performs the PCA, SVM and KNN.
  }
  #pragma omp section
  {
      In this section the thread 1 manages the
      Device 1 and performs the K-means.
  }
}
```

The execution is divided into two sections, each one assigned to a different thread. Each GPU is associated to a thread through the function *cudaSetDevice*. In this way, the *thread 0* (Section 0) manages the *Device 0* and *thread 1* (Section 1) manages the *Device 1*. The CPU-threads work in parallel to compute simultaneously the different algorithms presented in the two previous sections. On the one hand, the *thread 0* manages the PCA, SVM and KNN computation in the same way described for the *single-GPU* code: the PCA is executed on the *Device 0*, except for the eigenvectors computation executed on the host. The SVM is also executed on the same GPU and its output, together with the PCA output, is sent to the KNN, also executed on this device. The KNN output is transferred to the host through a *cudamemcpy* function managed by the *thread 0* (red dashed lines from KNN to Majority Voting in Fig. 4). On the other hand, the *thread 1* computes the K-means *Clusters initialization* on the host, transferring then the clusters centroids to the *Device 1* where the other steps of the K-means are computed. At the end, the output is transferred to the host (red dashed lines from K-means to Majority Voting in Fig. 4). At this point the parallel section is closed and the computation is managed only by one thread. The Majority Voting is performed on the host and the new classified image is displayed. It is worth noting that the algorithm division among the devices, performed in the multi-GPU versions, is based on data dependencies.

3) MULTI-GPU PARALLEL VERSION 2 (CS-MULTI2)

The second *multi-GPU* version is called *CS-Multi2* and it differs from the previous one in the point where the parallel

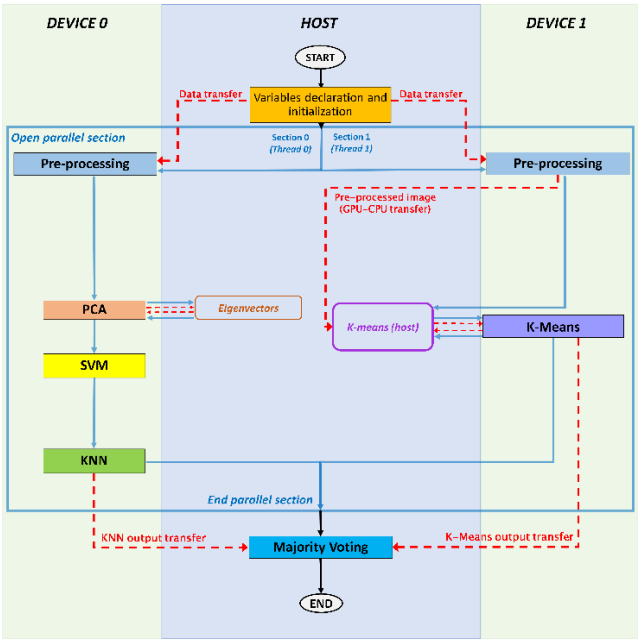

FIGURE 5. Flow diagram of the parallel code of the multi-GPU parallel version 2 (*CS-multi2*) of the complete HS brain cancer detection algorithm.

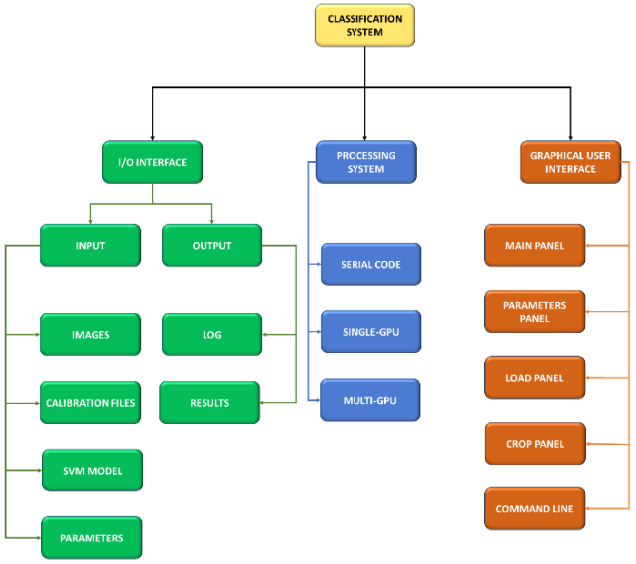

FIGURE 6. Hierarchical pattern tree of the brain classification system.

section is generated. As it is possible to see in Fig. 5, the variables declaration and initialization are always performed on the host. After this phase, the parallel section is created in the same way described in Algorithm A. The main difference compared to the *CS-Multi1* is that, since the pre-processing computation is performed in both devices, each GPU requires the initial data. For this reason, there is a double transfer of the initial data, one managed by the *thread 0* for the Section 0 and the other one by the *thread 1* for the Section 1. The code in each section can be executed in parallel: each thread manages the data transfer to its device where the pre-processing phase can start. After this step, in Section 0, the PCA, SVM and KNN are computed as explained before. In Section 1, the pre-processed image is transferred from the *Device 1* to the host in order to compute the first part of the K-means. Then the computation continues on the device as described above.

In this case, there is no need of a GPU-GPU transfer since each device computes a pre-processing phase, so the pre-processed image is already stored in the global memory of each GPU.

E. GRAPHICAL USER INTERFACE (GUI)

In order to manage in a user friendly way the different system versions developed in this work, a specific GUI was developed to be installed in the HS intraoperative demonstrator for real-time processing of the captured data in surgical brain procedures. The source code implementation of the GUI has been written in C/C++ language. The project is set up in a hierarchical structure (Fig. 6), identifying three main modules: *I/O Interface*, *Processing System*, and *Graphical User Interface*.

The *I/O Interface* provides a communication channel between the user and the system, allowing the information exchange. In particular, the *Input* channel includes all the classifiers inputs and parameters divided in five components. The *Output* part contains the classification maps (*Results*) and a *.txt* file with all the main information about the execution (*Log*). The *Processing System* module presents three main components, each one related to the platform used in the elaboration: the *Serial* version, the *single-GPU* and the *multi-GPUs* codes. The *Graphical User Interface*module manages five interaction screens. The *main panel* is shown in Fig. 7.A, where in the top-left side three buttons have been placed (to load the image, to change the parameters and to start the classification) and a drop-down list, which shows the types of computation that the user can choose on the base of the technology that he/she wants to exploit. The system is flexible enough to be executed on different platforms. For this reason, the user can select a serial computation performed by a CPU or a parallel computation performed by a single or a multi-GPU technology. Pressing the *load* button, the user opens the *load panel* where it is possible to select the HS image to classify. If the *parameters* button is pressed, the window shown in Fig. 7.C is opened (*parameters panel*). This window presents all the parameters of the different algorithms described above with the default values. The user can choose to change these parameters and try new system configurations. After choosing the new system configuration, the user can upload the image (Fig. 7.B) and decide whether to classify the whole image or only a part of it (*crop panel*). When the button *process* is pressed, the computation starts. The classification result is shown in the right part of the *main panel* (Fig.7.A). Finally, the *command line* shows the execution status.

Typically, a system like the one presented requires a tool that provides an excellent usability even to those who are not

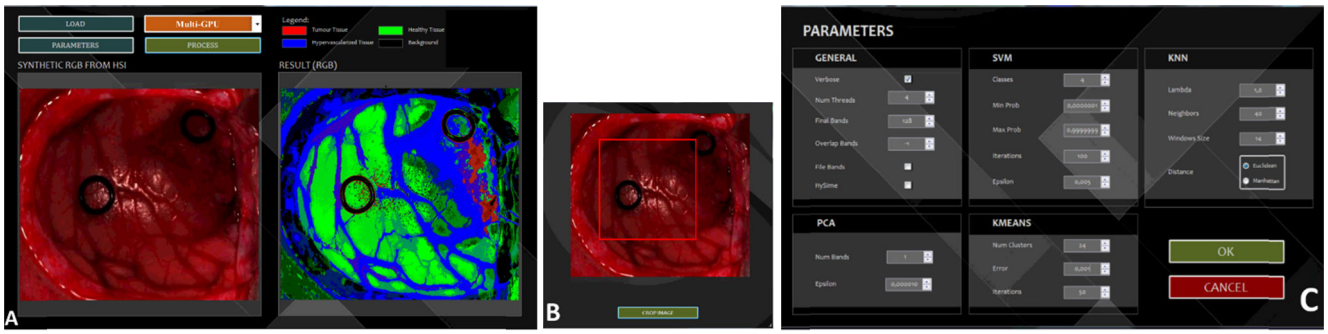

FIGURE 7. Graphical User Interface of the HS intraoperative demonstrator. A) Main panel; B) Crop panel; C) Parameter configuration panel.

familiar with computer science details. Therefore, the GUI has been developed following the surgeons' suggestions in order to provide an ease of use and intuitive interface. As an example, the dark theme was chosen since it is more suitable for the surgical room. During the neurosurgery, the user has only to follow three simple steps to start the classification (i.e., load the image, select the region of interest and press the *process* button). For this reason, during the surgical procedure, the user mainly works with the panel in Fig. 7.A. The *parameters panel* (Fig. 7.C) is mainly used for research purposes and not during the tumor resection. This GUI is currently used in the demonstrator located at the Doctor Negrin Hospital of Las Palmas de Gran Canaria.

III. RESULTS AND DISCUSSION

The development of an optimized version of the brain cancer detection system required several steps and analysis before reaching an efficient version capable of satisfying the real-time constraint. As described in the previous sections, the first step of the work has been the development of the optimized versions of the algorithms that compose the HS brain cancer detection system.

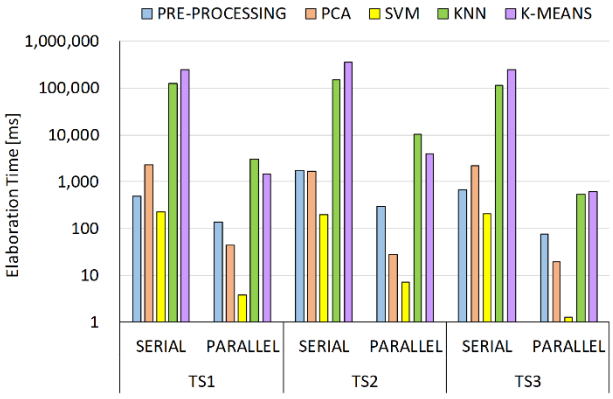

FIGURE 8. Average processing times of the single algorithms exploiting the TS1, TS2 and TS3 systems. Time shown in milliseconds and in logarithmic scale.

The chart presented in Fig. 8 shows the average processing times of the serial and parallel algorithms exploiting the three test systems. For each of them, it is possible to notice that the KNN and the K-means are the algorithms with the highest computational load, while the SVM is the faster one. The main aspect that should be noticed is that all the parallel versions present a considerable reduction of the execution times, which facilitates the achievement of a real-time elaboration. The TS3 system reaches the best results if the parallel codes are considered.

Several strategies and implementations have been described above but, according to the results shown in Fig. 8, it is possible to conclude that the GPU technology, together with an efficient design of the algorithm codes, allows reaching high speed-up factors and saving computational time. Right after the choice of the best solution for each algorithm, it has been studied the most efficient way to gather all these versions together in order to develop a highly optimized hybrid classification system, exploiting both single and multi-GPU technologies. Table 4 shows the comparison between the computational times of the parallel versions and the serial ones, presenting also the obtained speed-up.

TABLE 4. Serial and parallel processing times and speed-up.

System ID and Implementation		Image ID				
		P1C1	P1C2	P2C1	P3C1	P4C1
TS1	Serial [s]	560.28	709.11	514.62	469.17	291.12
	CS-Single [s]	18.54 (30.22×)	20.95 (33.85×)	15.51 (33.16×)	13.33 (35.20×)	6.36 (45.80×)
	CS-Multi1 [s]	14.94 (37.51×)	15.99 (44.34×)	11.99 (42.90×)	10.44 (44.95×)	4.03 (72.33×)
	CS-Multi2 [s]	14.7 (38.11×)	15.74 (45.05×)	11.79 (43.63×)	10.38 (45.18×)	3.84 (75.81×)
TS2	Serial [s]	463.52	539.48	367.50	350.79	228.76
	CS-Single [s]	6.61 (70.14×)	7.33 (73.64×)	5.55 (66.21×)	4.92 (71.27×)	2.27 (100.88×)
TS3	Serial [s]	387.46	481.66	353.42	313.53	179.28
	CS-Single [s]	2.68 (144.77×)	2.82 (170.74×)	**2.29** (**154.04**×)	**2.25** (**139.14**×)	**1.68** (**106.54**×)
	CS-Multi1 [s]	**2.44** (**158.51**×)	**2.67** (**180.31**×)	2.45 (143.94×)	2.49 (125.85×)	2.01 (89.36×)
	CS-Multi2 [s]	2.56 (151.17×)	2.68 (179.86×)	2.40 (147.07×)	2.31 (135.69×)	2.02 (88.74×)

Best result for each image has been highlighted in boldface. The different versions have been tested on three different processing systems that have been presented in Section II.B. TS1 and TS3 are multi-GPU systems, containing two GPU boards each one. For this reason, they have been exploited to test both the single and multi-GPU versions.

An analysis of the results shown in Table 4 reveals that all the parallel versions classify the HS images in less than one minute, meeting the real-time constraint for this intraoperative application. As far as the serial processing times are concerned, they can take more than 10 minutes to classify the images in some cases. Indeed, a serial code profiling shows that the KNN and the K-means algorithms have the higher computational loads, causing the non-compliance of the real-time constraint. In fact, the KNN takes the 20% - 39% of the total processing time (considering all the images and all the systems) while the K-means takes the 55% - 86% of the total time. On the other hand, the pre-processing, PCA and SVM are very efficient algorithms considering that the processing of each of them takes less than the 1% of the total time.

Analyzing the speed-up obtained between the *CS-Single* versions and their corresponding serial codes, it can be noticed that the best performing device is the NVIDIA RTX 2080 (TS3), achieving a maximum speed-up factor of about 170×. This result is obtained due to the more efficient recent architecture, the highest clock frequency and the largest number of CUDA cores that characterize this board. It is also interesting to notice that the NVIDIA GTX 1060 board (TS2) reaches higher speed-up than the NVIDIA Tesla K40 (TS1), even if it has lower number of cores. On the other hand, the GTX 1060 board benefits from a more recent architecture and a higher working frequency, which allows it to reach speed-up factors of about 100×. The consistent reduction of the *CS-Single* classification times is mainly due to the efficient parallelization of the most consuming algorithms (KNN and K-means). In fact, in this first parallel version, the KNN takes only 0.71 s compared to the 161.26 s of the serial code, when the biggest image (worst case) and TS3 are considered. In addition, the K-means algorithm presents a meaningful processing time reduction: the parallel clustering takes 0.88 s instead of 326.72 s in the serial code (always considering the biggest image and the TS3). Despite the pre-processing, PCA and SVM are developed with very efficient serial codes. Their processing times have been also reduced in the parallel version, increasing the speed-up factor of the entire classification. It is worth noticing that the *CS-Single* version performs a very fast classification, elaborating the biggest image of the database (P1C2) in only 2.82 s when the TS3 is used.

Despite the single-GPU version allows a significant reduction of the computational times of the serial code, the most performant parallel solutions are the ones that exploit the multi-GPU technology. Thus, in this case, the supervised and the unsupervised classifications are distributed on two different GPUs, executing the two most consuming time algorithms, KNN and K-means, on the two boards simultaneously. The obtained gain in the total classification time is strictly related to the K-means performance, since this is the algorithm moved to the second GPU and elaborated simultaneously to the KNN.

Analyzing Table 4, it is possible to see that the gap between the *CS-Single* and the *CS-Multi1/2*, elaborated with the TS3, is lower than the one obtained with the TS1. The reason is mainly the different K-means processing times obtained with the NVIDIA RTX 2080 and the NVIDIA Tesla K40 GPUs. As said before, considering the P1C2 image, the RTX 2080 takes only 0.88 s to process the K-means, compared to the 5.31 s of the Tesla K40. For this reason, there is a higher time decrease between the single and multi-GPU versions in TS1 than in TS3.

Fig. 9 presents several chronograms, where the algorithm processing times in the *CS-Single* and *CS-Multi1* versions elaborated by the TS1 (Fig. 9.A and B) and by the TS3 (Fig. 9.C and D) are detailed. It is possible to consider that the K-means bar follows the KNN one in the *CS-Single* graphs (Fig. 9.A and C), while in the *CS-Multi1* graphs they are overlapped since the computation is performed by two GPUs simultaneously. Moving the K-means to a second board allows obtaining a temporal gain (red arrows in Fig. 9) which is higher in TS1 than in TS3 for the reason explained before. Despite this, TS3 achieves the fastest classification since it features a more efficient algorithms parallelization than TS1. Thus, TS3 takes only 2.67 s to classify the image P1C2 (*CS-Multi1*), while TS1 takes 15.99 s.

Analyzing the results of the TS3 system, another consideration that can be done is that for the three smallest images of the database (P2C1, P3C1 and P4C1) the multi-GPU algorithms are slightly slower than the single-GPU. This is mainly due to the OpenMP context creation and management, present in the multi-GPU versions, which affect most the performance of the smallest images.

In addition, data reported in Table 4 highlight that the two multi-GPU versions results are not significantly different. This means that, for this type of systems, performing two pre-processing steps (one in each GPU) takes almost the same time than computing only one pre-processing and transferring the pre-processed image through a GPU-GPU data transfer. In fact, the profiling of the parallel code shows that data transfers are not so relevant in the execution. The last consideration that can be done analyzing the results in Table 4 is that TS2 and TS3 perform the fastest parallel classification, and this is due to their most recent GPUs, which work at higher frequencies.

In conclusion, it is possible to affirm that all the parallel codes can elaborate a real-time classification for all the images, taking only few seconds in the classification. This result highlights that the desktop-based system equipped with one or two GPUs is the most suitable technology to introduce in the operating room. In fact, in addition to allow a local processing of the data, it is capable of classifying the images thoroughly satisfying the 1-minute constraint.

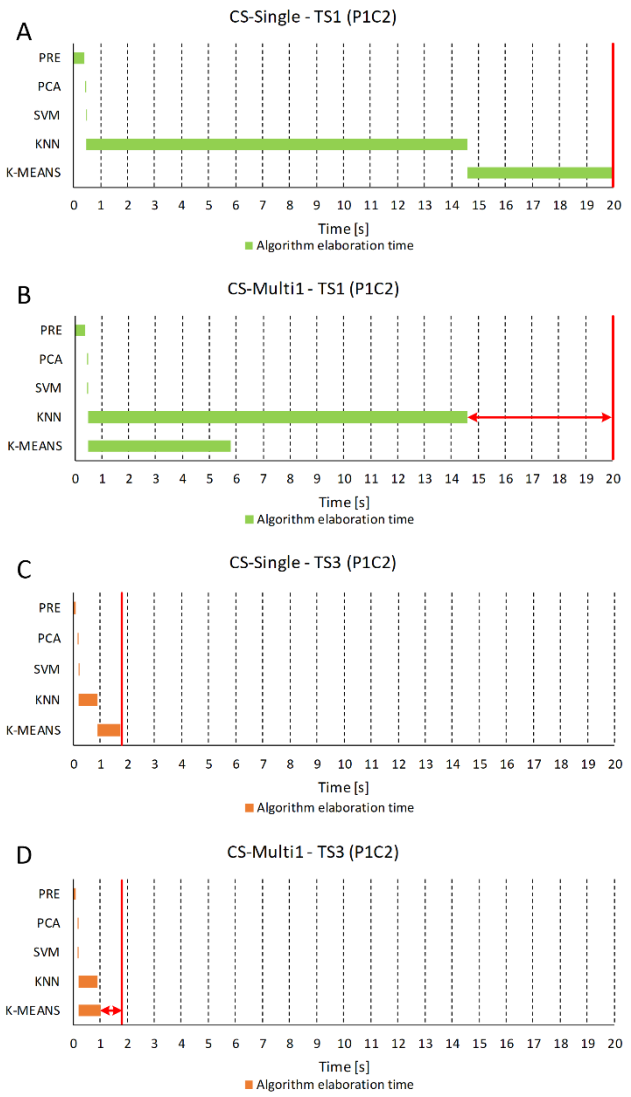

FIGURE 9. Algorithms processing times in the P1C2 classification (worst case). A-B) Times related to the *CS-Single* and *CS-Multi1* versions computed by the TS1. C-D) Times related to the *CS-Single* and *CS-Multi1* versions computed by the TS3. The graphs report only the algorithm times and durations, not considering the ones related to the memory allocations and deallocations, CUDA contest creation and data transfers.

A. QUALITATIVE EVALUATION OF THE RESULTS

The parallel versions of the brain cancer detection system elaborate the same classification maps already presented in [12]. Fig. 10 shows the classification map related to the image P2C1 (Fig. 10.F), reporting also all the partial algorithm results. Fig. 10.A shows an RGB representation of the HS image, i.e. the input of the supervised and unsupervised classifications. Once the PCA algorithm is applied, the one-band-representation of the HS image is generated (Fig. 10.B), where it is possible to notice the spatial representation of the most relevant spectral information after the data dimensionality reduction. On the other hand, the classification result of the SVM algorithm is shown in Fig. 10.C. This map associates a specific color to each class, displaying

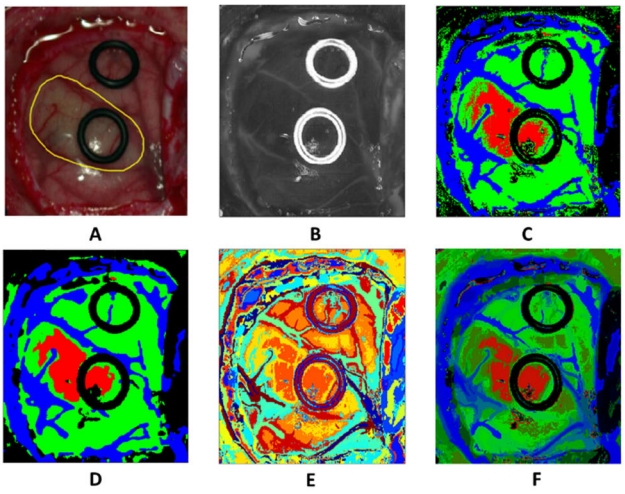

FIGURE 10. A) RGB representation of the HS image; the yellow line indicates the tumor location; B) One-band-representation (PCA output); C) Classification map (SVM output); D) Filtered classification map (KNN output); E) Unsupervised segmentation map (K-means output); F) Final classification map (Majority Voting output).

the tumor area in red, the normal brain tissue in green, the hypervascularized tissue in blue (mainly blood vessels) and the background (other substances and materials presented in a brain surgical scene) in black.

Fig. 10.B and C are the inputs of the KNN filtering algorithm that performs a spatial homogenization of the classification map. The result of the KNN step is shown in Fig. 10.D. While this map differentiates each class from the others on the base of the histological meaning, the K-means map (Fig. 10.E) represents several clusters with different colors without any physiological meaning, in terms of interpreting the results. This kind of clustering allows delineating with high accuracy the boundaries of some biological structures such as blood vessels, materials like the ring markers and several different tissues. Even if the clusters have not a histological meaning, the different spectral regions are well delimited. This result is consistent with the main goal of the system that is to delineate the tumor area with high accuracy. The final classification map (Fig. 10.F) is generated with the Majority Voting algorithm which merges the supervised and unsupervised classification outputs (Fig. 10.E and F) obtaining a very accurate representation of the tumor where the boundaries are precisely delimited.

B. COMPARISON WITH OTHER WORKS

Originally, HS images were used in remote sensing applications in the military area. The recent technological progress allowed a more widespread use of the HS images also in other fields, such as medicine and, in particular, cancer detection. In the state of the art, HSI was used in different medical applications for the analysis of several types of cancer, such as cervix [34], breast [35], skin [36], stomach [37], prostate [38], tongue [39] and brain cancer [21]. Most of these works focus on providing a methodology for the HSI classification. Only [21], [37], [39] report a technological implementation

of the proposed solutions. In [37], the authors performed a combination of SVM and Spectral Standard Deviation in order to provide an index to discriminate between cancerous and healthy tissues. The system acquires an image in 15 s and computes the index in less than 20 s. The processing was performed on a computer working at 3.6 GHz with a 2 GB RAM and running in MATLAB environment. It is not fair to compare this system with the one presented in this work mainly because it is based on a simpler method. Despite this, it is important to highlight that this solution does not provide a real-time classification. In [39], the authors proposed a sparse representation to detect the tumor in the tongue. The system exploits an Intel® Core™i7 CPU equipped with 4 GB of RAM and analyzes the image in 3.4 s. In this case, it is not possible to make a fair comparison since authors do not declare if the elaboration is in real-time or not. However, the only consideration that can be done is that it contains only one algorithm, whose processing takes longer than our best result where a combination of several algorithms is employed. A fairer comparison can be made between our work and the one presented in [21]. This work was developed within the same project where this research is framed and, for this reason, it adopts the same classification algorithm presented in this paper. In fact, one aim of the HELICoiD project was the evaluation of different technologies in order to select the most efficient one for real-time brain cancer detection. In [21], the classification system has been developed on a Kalray Massively Parallel Processor Array (MPPA®) combined with a Intel® Core™i7-4770k 3.5 GHz equipped with 8 GB of RAM. This solution provides a large speedup compared to serial processing, achieving a classification time of about 1 minute (near real-time). It is worth noticing that an efficient code for GPU technology provides a faster classification than the Kalray solution. On the other hand, it is important to highlight that the Kalray technology is optimized for low power embedded applications. In this case, there are no power restrictions in a surgical environment, thus, the GPU device a more suitable technology for real-time classification in this application. To the best of the authors' knowledge, in the state of the art, there are some HS classification systems developed for remote sensing applications [7], [40]–[42]. However, it is not possible to make a fair comparison between these works and the present solution mainly because they are based on other kinds of techniques and restrictions, such as power consumption. The proposed solution has been developed in order to solve a specific problem for intraoperative brain cancer resection. In this application, it is of crucial importance to have a real-time classification system capable of delineating the cancer borders with high accuracy, in order to facilitate the surgeon's actions during tumor resection.

IV. CONCLUSION

The work presented in this paper developed several parallel versions of an HS brain cancer detection system capable of performing a real-time classification of HS images. During a neurosurgical operation, the system is able to provide useful information to the surgeon indicating the location and delineation of the tumor area in order to facilitate its resection.

The parallel versions of the system presented in this work can provide the result in real-time. The time constraint has been set to 1 minute, i.e. the time taken by the camera to acquire an HS image. The fastest parallel version described in this work takes only 2.67 s to elaborate the largest image of the database, thoroughly satisfying the real-time constraint. Moreover, all the parallel versions are real-time compliant, classifying the images in less than 21 seconds. This result has been reached exploiting HPC technologies and, in particular, hybrid systems equipped with the most recent multicore CPUs and manycore GPUs.

The experimental results demonstrated that the use of GPUs allows reaching high performances in those applications where a huge amount of data have to be elaborated in a short period of time. The use of the HS images for brain cancer detection is a representative example of this kind of applications, due to the large amount of data that characterizes the HS images. Moreover, a real-time classification is of critical importance considering that this system is used during neurosurgical procedures. Taking into account that the operating theatres do not present power restrictions, the main drawback of the GPU technology is overcome. In addition, the choice of a monolithic system allows to locally process data, avoiding data transmission to remote technological solutions. In this kind of applications, having an isolated system is of crucial importance due to security and ethical reasons in the usage of sensitive data. Finally, the results obtained in this work show a great potential in the use of GPUs for processing HS data intraoperatively in real-time, which will allow in the near future the real-time processing of hyperspectral video imaging.

ACKNOWLEDGMENT

The authors would like to thank NVIDIA Corporation for the donation of the NVIDIA Tesla K40 GPU used for this research. Additionally, this work was completed while Samuel Ortega was beneficiary of a pre-doctoral grant given by the "*Agencia Canaria de Investigacion, Innovacion y Sociedad de la Información (ACIISI)*" of the "*Conserjería de Economía, Industria, Comercio y Conocimiento*" of the "*Gobierno de Canarias*", which is part-financed by the European Social Fund (FSE) (*POC 2014-2020, Eje* 3 *Tema Prioritario 74 (85%)*).

REFERENCES

[1] M. Halicek, H. Fabelo, S. Ortega, G. M. Callico, and B. Fei, "*In–Vivo* and *Ex–Vivo* tissue analysis through hyperspectral imaging techniques: Revealing the invisible features of cancer," *Cancers*, vol. 11, no. 6, p. 756, May 2019.

[2] G. Lu and B. Fei, "Medical hyperspectral imaging: A review," *J. Biomed. Opt*, vol. 19, no. 1, Jan. 2014, Art. no. 010901.

[3] S. Ortega, "Use of hyperspectral/multispectral imaging in gastroenterology. shedding some-different-light into the dark," *J. Clin. Med.*, vol. 8, no. 1, p. 36, Jan. 2019.

[4] M. Kamruzzaman and D.-W. Sun, "Introduction to hyperspectral imaging technology," in *Computer Vision Technology for Food Quality Evaluation*. Amsterdam, The Netherlands: Elsevier, Jan. 2016, pp. 111–139.

[5] T. Adão, J. Hruška, L. Pádua, J. Bessa, E. Peres, R. Morais, and J. Sousa, "Hyperspectral imaging: A review on UAV–based sensors, data processing and applications for agriculture and forestry," *Remote Sens.*, vol. 9, no. 11, p. 1110, Oct. 2017.
[6] E. Torti, A. Fontanella, A. Plaza, J. Plaza, and F. Leporati, "Hyperspectral image classification using parallel autoencoding diabolo networks on multi–core and many–core architectures," *Electronics*, vol. 7, no. 12, p. 411, Dec. 2018.
[7] Z. Wu, L. Shi, J. Li, Q. Wang, L. Sun, Z. Wei, J. Plaza, and A. Plaza, "GPU parallel implementation of spatially adaptive hyperspectral image classification," *IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens.*, vol. 11, no. 4, pp. 1131–1143, Apr. 2018.
[8] Z. Wu, Q. Wang, A. Plaza, J. Li, J. Wei, and Z. Wei, "GPU implementation of hyperspectral image classification based on weighted Markov random fields," in *Proc. 8th Workshop Hyperspectral Image Signal Process., Evol. Remote Sens. (WHISPERS)*, 2017.
[9] E. Torti, A. Fontanella, G. Florimbi, F. Leporati, H. Fabelo, S. Ortega, and G. Callico, "Acceleration of brain cancer detection algorithms during surgery procedures using GPUs," *Microprocessors Microsyst.*, vol. 61, pp. 171–178, Sep. 2018.
[10] E. Torti, G. Florimbi, F. Castelli, S. Ortega, H. Fabelo, G. Callicó, M. Marrero-Martin, and F. Leporati, "Parallel K–Means Clustering for Brain Cancer Detection Using Hyperspectral Images," *Electronics*, vol. 7, no. 11, p. 283, Oct. 2018.
[11] H. Fabelo *et al.*, "*In–Vivo* hyperspectral human brain image database for brain cancer detection," *IEEE Access*, vol. 7, pp. 39098–39116, 2019.
[12] H. Fabelo *et al.*, "Spatio-spectral classification of hyperspectral images for brain cancer detection during surgical operations," *PLoS ONE*, vol. 13, no. 3, Mar. 2018, Art. no. e0193721.
[13] N. Sanai and M. S. Berger, "Glioma extent of resection and its impact on patient outcome," *Neurosurgery*, vol. 62, no. 4, pp. 753–766, Apr. 2008.
[14] N. Sanai and M. S. Berger, "Operative techniques for gliomas and the value of extent of resection," *Neurotherapeutics*, vol. 6, no. 3, pp. 478–486, Jul. 2009.
[15] K. Petrecca, M.-C. Guiot, V. Panet-Raymond, and L. Souhami, "Failure pattern following complete resection plus radiotherapy and temozolomide is at the resection margin in patients with glioblastoma," *J. Neuro-Oncol.*, vol. 111, no. 1, pp. 19–23, Jan. 2013.
[16] I. J. Gerard, M. Kersten-Oertel, K. Petrecca, D. Sirhan, J. A. Hall, and D. L. Collins, "Brain shift in neuronavigation of brain tumors: A review," *Med. Image Anal.*, vol. 35, pp. 403–420, Jan. 2017.
[17] M. Knauth, C. R. Wirtz, V. M. Tronnier, N. Aras, S. Kunze, and K. Sartor, "Intraoperative MR imaging increases the extent of tumor resection in patients with high-grade gliomas," *Amer. J. Neuroradiol.*, vol. 20, no. 9, pp. 1642–1646, 1999.
[18] A. G. Chacko, N. K. S. Kumar, G. Chacko, R. Athyal, V. Rajshekhar, and G. Unsgaard, "Intraoperative ultrasound in determining the extent of resection of parenchymal brain tumours–a comparative study with computed tomography and histopathology," *Acta Neurochirurgica*, vol. 145, pp. 743–748, Sep. 2003.
[19] F. W. Floeth, M. Sabel, C. Ewelt, W. Stummer, J. Felsberg, G. Reifenberger, H. J. Steiger, G. Stoffels, H. H. Coenen, and K.-J. Langen, "Comparison of ^{18}F-FET PET and 5-ALA fluorescence in cerebral gliomas," *Eur. J. Nucl. Med. Mol. Imag.*, vol. 38, no. 4, pp. 731–741, Apr. 2011.
[20] W. Stummer, U. Pichlmeier, T. Meinel, O. D. Wiestler, F. Zanella, and H.-J. Reulen, "Fluorescence-guided surgery with 5-aminolevulinic acid for resection of malignant glioma: A randomised controlled multicentre phase III trial," *Lancet oncol.*, vol. 7, no. 5, pp. 392–401, May 2006.
[21] H. Fabelo *et al.*, "An intraoperative visualization system using hyperspectral imaging to aid in brain tumor delineation," *Sensors*, vol. 18, no. 2, p. 430, Feb. 2018.
[22] G. Florimbi, H. Fabelo, E. Torti, R. Lazcano, D. Madroñal, S. Ortega, R. Salvador, F. Leporati, G. Danese, A. Báez-Quevedo, G. Callicó, E. Juárez, C. Sanz, and R. Sarmiento, "Accelerating the K–nearest neighbors filtering algorithm to optimize the real–time classification of human brain tumor in hyperspectral images," *Sensors*, vol. 18, no. 7, p. 2314, Jul. 2018.
[23] H. Fabelo, S. Ortega, R. Guerra, G. Callicó, A. Szolna, J. F. Piñeiro, M. Tejedor, S. López, and R. Sarmiento, " A novel use of hyperspectral images for human brain cancer detection using *in-Vivo* samples," in *Proc. 9th Int. Joint Conf. Biomed. Eng. Syst. Technol.*, 2016, pp. 311–320.
[24] I. Guides. (2018). *CUDA Toolkit Documentation v10.0.130*. [Online]. Available: http://docs.nvidia.com/cuda/
[25] NVIDIA. *cuBLAS Library Documentation*. Accessed: May 9, 2018. https://docs.nvidia.com/cuda/cublas/index.html
[26] R. Lazcano, D. Madroñal, R. Salvador, K. Desnos, M. Pelcat, R. Guerra, H. Fabelo, S. Ortega, S. Lopez, G. Callico, E. Juarez, and C. Sanz, "Porting a PCA-based hyperspectral image dimensionality reduction algorithm for brain cancer detection on a manycore architecture," *J. Syst. Archit.*, vol. 77, pp. 101–111, Jun. 2017.
[27] NVIDIA. *cuSOLVER Library Documentation v10.0.130*. Accessed: Sep. 10, 2019. [Online]. Available: https://docs.nvidia.com/cuda/cusolver/index.html
[28] D. Madroñal, R. Lazcano, R. Salvador, H. Fabelo, S. Ortega, G. Callico, E. Juarez, and C. Sanz, "SVM-based real-time hyperspectral image classifier on a manycore architecture," *J. Syst. Archit.*, vol. 80, pp. 30–40, Oct. 2017.
[29] C. Chang and C. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, p. 27, May 2013.
[30] K. Huang, S. Li, X. Kang, and L. Fang, "Spectral–spatial hyperspectral image classification based on KNN," *Sens. Imag.*, vol. 17, no. 1, pp. 1–13, 2016.
[31] Y. Tarabalka, J. A. Benediktsson, and J. Chanussot, "Spectral–spatial classification of hyperspectral imagery based on partitional clustering techniques," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, no. 8, pp. 2973–2987, Aug. 2009.
[32] OpenMP. *The OpenMP API Specification for Parallel Programming*. Accessed: Sep. 10, 2019. [Online]. Available: https://www.openmp.org/
[33] NVIDIA. *NVIDIA GPUDirect*. Accessed: Sep. 10, 2019. [Online]. Available: https://developer.nvidia.com/gpudirect
[34] R. H. Wilson and M.-A. Mycek, "Models of light propagation in human tissue applied to cancer diagnostics," *Technol. Cancer Res. Treatment*, vol. 10, no. 2, pp. 121–134, Apr. 2011.
[35] S. V. Panasyuk, S. Yang, D. V. Faller, D. Ngo, R. A. Lew, J. E. Freeman, and A. E. Rogers, "Medical hyperspectral imaging to facilitate residual tumor identification during surgery," *Cancer Biol. Therapy*, vol. 6, no. 3, pp. 439–446, Mar. 2007.
[36] L. A. Zherdeva, "Hyperspectral imaging of skin and lung cancers," *Proc. SPIE Biophoton., Photon. Solutions for Better Health Care V*, vol. 9887, Apr. 2016, Art. no. 98870S.
[37] H. Akbari, K. Uto, Y. Kosugi, K. Kojima, and N. Tanaka, "Cancer detection using infrared hyperspectral imaging," *Cancer Sci.*, vol. 102, no. 4, pp. 852–857, Apr. 2011.
[38] H. Akbari, L. V. Halig, D. M. Schuster, A. Osunkoya, V. Master, P. T. Nieh, G. Z. Chen, and B. Fei, "Hyperspectral imaging and quantitative analysis for prostate cancer detection," *J. Biomed. Opt*, vol. 17, no. 7, Jul. 2012, Art. no. 0760051.
[39] Z. Liu, H. Wang, and Q. Li, "Tongue tumor detection in medical hyperspectral images," *Sensors*, vol. 12, no. 1, pp. 162–174, Dec. 2011.
[40] E. Torti, G. Danese, F. Leporati, and A. Plaza, "A hybrid CPU-GPU real-time hyperspectral unmixing chain," *IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens.*, vol. 9, no. 2, pp. 945–951, Feb. 2016.
[41] M. E. Paoletti, J. Haut, J. Plaza, A. Plaza, Q. Liu, and R. Hang, "Multicore implementation of the multi-scale adaptive deep pyramid matching model for remotely sensed image classification," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Jul. 2017, pp. 2247–2250.
[42] M. Paoletti, J. Haut, J. Plaza, and A. Plaza, "A new deep convolutional neural network for fast hyperspectral image classification," *ISPRS J. Photogram. Remote Sens.*, vol. 145, pp. 120–147, Nov. 2018.

GIORDANA FLORIMBI was born in Teramo, Italy, in 1989. She received the bachelor's degree in biomedical engineering from Università Politecnica delle Marche, Ancona, Italy, in 2012, and the master's degree in bioengineering and the Ph.D. degree in bioengineering and bioinformatics from the University of Pavia, Pavia, Italy, in 2015 and 2019, respectively. She is currently a Postdoctoral Researcher with the Engineering Faculty, University of Pavia. Her research is focused in real-time elaborations in the support medical systems development and in neuroscience, and exploiting HPC technologies.

HIMAR FABELO received the degree in telecommunication engineering and the M.Sc. and Ph.D. degrees in telecommunication technologies from the University of Las Palmas de Gran Canaria (ULPGC), Las Palmas de Gran Canaria, Spain, in 2013, 2014, and 2019, respectively. Since then, he has conducted his research activity with the Integrated System Design Division, Institute for Applied Microelectronics (IUMA), ULPGC, in the field of electronic and bioengineering. In 2015, he started to work as a Co-Ordination Assistant and a Researcher in the HELICoiD European Project, co-funded by the European Commission. In 2018, he performs a research stay with the Department of Bioengineering, the Erik Jonsson School of Engineering and Computer Science, The University of Texas at Dallas (UTD), collaborating with Prof. B. Fei in the field of medical hyperspectral imaging analysis using deep learning. His research interests include the use of machine learning and deep learning techniques applied to hyperspectral images to discriminate between healthy and tumor samples for human brain tissues in real-time during neurosurgical operations.

EMANUELE TORTI was born in Voghera, Italy, in 1987. He received the bachelor's degree in electronic engineering, the master's degree *(cum laude)* in computer science engineering, and the Ph.D. degree in electronics and computer science engineering from the University of Pavia, Pavia, Italy, in 2009, 2011, and 2014, respectively. He is currently an Assistant Professor with the Engineering Faculty of the University of Pavia. His research is focused on high-performance architectures for real-time image processing and signal elaboration.

SAMUEL ORTEGA received the degree in telecommunication engineering and the M.Sc. degree in telecommunication technologies from the University of Las Palmas de Gran Canaria (ULPGC), Spain, in 2014 and 2015, respectively. Since then, he has conducted his research activity with the Integrated System Design Division, Institute for Applied Microelectronics (IUMA), ULPGC, in the field of electronic and bioengineering. In 2015, he started to work as a Co-Ordination Assistant and a Researcher with the HELICoiD European Project, co-funded by the European Commission. His current research interests are in the use of machine learning algorithms in medical applications using hyperspectral images.

MARGARITA MARRERO-MARTIN received the M.S. degree in telecommunication engineering and the Ph.D. degree from the University of Las Palmas de Gran Canaria (ULPGC), Spain, in 2001 and 2012, respectively. She is currently an Associate Professor with ULPGC. She researches with the Institute for Applied Microelectronics involved in the Microelectronics Technology Division. Until 2017, her research lines of interest were the characterization, modeling, and design of RF integrated passive components. She has participated as a Researcher in national and European funded projects, coauthored more than 50 articles in journal articles and conferences papers. Since 2017, she has been changed her field of research to the area of bioengineering, specifically the use of hyperspectral imaging to detect cancer in real time. She has occupied different management positions at ULPGC.

GUSTAVO M. CALLICO (Member, IEEE) received the M.S. degree (Hons.) in telecommunication engineering, the Ph.D. degree (Hons.), and the European Doctorate (Hons.) from the University of Las Palmas de Gran Canaria (ULPGC), in 1995 and 2003, respectively. From 1996 to 1997, he was granted with a research grant from the National Educational Ministry. In 1997, he was hired by the university as an Electronic Lecturer. In 1994, he joined the Institute for Applied Microelectronics (IUMA). From 2000 to 2001, he stayed at the Philips Research Laboratories (NatLab), Eindhoven, The Netherlands, as a Visiting Scientist, where he developed his Ph.D. thesis. He is currently an Associate Professor with ULPGC. He also develops his research activities with the Integrated Systems Design Division, Institute for Applied Microelectronics (IUMA). He has more than 170 publications in national and international journals, conferences, and book chapters. He has participated in 18 research projects funded by the European Community, the Spanish Government, and international private industries. Since 2015, he has been the responsible for the scientific-technological equipment project called Hyperspectral image acquisition system of high-spatial and spectral definition, granted by the General Directorate of Research and Management of the National Research and Development Plan, funded through the General Directorate of Scientific Infrastructure. He has been the coordinator of the European Project HELICoiD Future and Emerging Technologies (FET), under the Seventh Framework Program with Grant Agreement 618080. He has been an invited Professor with the University of Pavia, Italy, in October 2015 and March 2019. His current research fields include hyperspectral imaging for real-time cancer detection (brain, skin, and cervix), real-time super-resolution algorithms, synthesis-based design for SOCs, and circuits for multimedia processing and video coding standards, especially for H.264 and SVC. He has been an Associate Editor of the IEEE TRANSACTIONS ON CONSUMER ELECTRONICS, since 2009, and IEEE ACCESS, since 2016. He is currently a Senior Associate Editor of the IEEE TRANSACTIONS ON CONSUMER ELECTRONICS.

GIOVANNI DANESE received the Ph.D. degree in electronics and computer engineering from the University of Pavia, Pavia, Italy, in 1987. He is currently a Full Professor of computer programming and computer architecture with the Engineering Faculty, University of Pavia. His research interests include parallel computing, computerized instrumentation special-purpose computers, and signal and image processing.

FRANCESCO LEPORATI received the Ph.D. degree in electronics and computer engineering from the University of Pavia, Pavia, Italy, in 1993. He is currently an Associate Professor of industrial informatics and embedded systems and digital systems design with the Engineering Faculty, University of Pavia. His research interests include automotive applications, FPGA and application specific-processors, embedded real-time systems, and computational physics. He is a member of the Euromicro Society and an Associate Editor of *Microprocessors and Microsystems*.