Hardware Parallel Architecture proposed to accelerate the Orthogonal Matching Pursuit Compressive Sensing Reconstruction

C. Osorio Quero*, D. Durini, R. Ramos-Garcia,
J. Rangel-Magdaleno, J. Martinez-Carranza.

National Institute of Astrophysics, Optics and Electronics (INAOE)
Luis Enrique Erro 1, 72840 Tonantzintla, Puebla, Mexico
*caoq@inaoep.mx

ABSTRACT

The compressive sensing (CS) technique is a novel tool used to reconstruct images using fewer samples, normally sparse in the transform domain, than those required by conventional imaging systems. However, the methods applied for signal reconstruction within the CS approach still present some problems in the implementation, mainly due to their intensive computational demand and high power consumption requirements. These drawbacks need addressing if this approach is followed in systems aimed at e.g. drone autonomous flying or other embedded applications that additionally require very short processing times. In this paper we evaluate the use of hardware based parallel processing architecture for the implementation of the Orthogonal Matching Pursuit (OMP) algorithm, one of the most efficient CS reconstruction algorithms developed so far. To improve the algorithm performance, we target different maximum allowed processing times to reach minimum image resolutions required by each system of interest using different sparse (16 and 64) amounts of single-pixel generated samples per image. We also target the final image resolution to be above 20 dB in terms of the peak signal-to-noise ratio (PSNR). To reduce the execution and processing times required to generate each image, we propose implementing parallel kernels in the hardware platform for each of the operations required by the algorithms under study. In the proposed implementation the reconstructed images are used to generate video streams that form the foundation on which decisions are to be made by the system in continuous time, whereby each single image (frame) reconstruction cannot overcome 30 ms in order to maintain the minimum amount of frames per second (fps) above 33 (minimum required for an acceptable video stream). The implementation of a variation of the OMP algorithm in a graphics processing unit (GPU) using parallel architecture approach allows obtaining processing times 4 or 5 times shorter than those obtained if central processing unit (CPU) based architecture implementation is used for the same purpose.

Keywords: 2D/3D Imaging, CUDA, GPU architecture, Orthogonal Matching Pursuit (OMP), patterns, parallel architecture, Compressive Sensing (CS)

1. INTRODUCTION

The compressive sensing (CS) technique [1] offers great accuracy for signal reconstruction applications when dealing with a reduced number of measured samples, expandable also to image reconstruction tasks. This approach offers shorter processing times [2] if compared to other currently used algorithms, allowing for lower energy consumption. Although the signal or image reconstruction processes may be complex, techniques have been developed, such as Orthogonal Matching Pursuit (OMP) [3], that allow the algorithms to get more efficient in what the minimum processing time required is concerned, and be easily adapted to different technology platforms such as Central Processing Units (CPU), Field Programmable Gate Arrays (FPGA) or Graphics Processing Units (GPU) [4]. The GPU architecture is oriented toward parallel operations and is a good candidate for the development of matrix based processing applications, normally used for image and video reconstruction. Recently, CS techniques have become the tool of choice for applications involving reconstruction of telecommunication signals, or images involved in processes ranging from Synthetic Aperture Radar (SAR) imaging [2] to single-pixel imaging (SPI) [5] used, for example, in biomedical applications. SPI is a perfect example of a system that requires image reconstruction based on an extremely

scarce amount of information, in this case a very reduced amount of illumination patterns generated by an array of illumination spots (individual LEDs or an illuminated array of micromirrors) sensed by a single pixel. Thus, we propose using the OMP algorithm implemented on both, the GPU and CPU technology platforms, respectively, for a comparative study dedicated to SPI applications. For the latter, we evaluated the OMP algorithm [2, 3], the algorithm based on the OMP using the Inverse Cholesky Factorization [6], and the Batch-OMP algorithm [7], respectively, concentrating specifically on the image reconstruction time and the peak signal-to-noise ratio (PSNR) of the reconstructed images. The performed evaluation allows to choose the most efficient algorithm to be implemented using the GPU platform in a parallelized manner for SPI tasks. After performing the first implementation using i5 CPU technology, a *Jetson Nano GPU* [8] was used for further implementation of the SPI image reconstruction. For the application of interest, the goal specification regarding the maximum processing time allowed for single-image reconstruction is of below 30 ms for reconstructed image sizes of respectively 64×64, 64×16, 128×16, and 256×16 virtual pixels. The maximum processing time defined will allow for video stream generation with a frame-rate (a frame being defined as each individual reconstructed image) close to 33 frames per second (fps) that should allow decision making in continuous time.

2. DEFINITION OF GOAL SPECIFICATIONS FOR THE SPI SYSTEM OF INTEREST

The SPI process time resolution is defined by two essential figures of merit. The first one is the time required to generate, project, and properly detect a series of different illumination patterns generated by e.g. an array of LEDs directed towards and finally reflected by the objects in the scene being imaged [5]. The amount of illumination patterns required to reconstruct a 2D image will depend on the spatial resolution (F_{min} in Eq. (1)) targeted in the application of interest. The minimum time needed to acquire a single illumination pattern by the photodetector (pixel) will be defined in terms of its optical sensitivity (that depends of its quantum efficiency and is normally expressed in terms of amperes per watt, A/W), its response time, and the distance and reflection index of the objects in the illuminated scene that reflect the light comprised within the illumination pattern that is finally detected by the photodetector. This reflected luminous signal must additionally at all times be stronger than the signal produced by the background illumination present in the system, and especially higher than the photon shot noise generated by that background illumination, in order to be properly detected and discriminated. The latter defines the minimum signal integration time (T_{int}) that the photodetector requires to sense each individual luminous pattern and consequently its minimum length of projection. The second figure of merit is the highest reachable frequency (F_{ADC} in Eq. (1)) of the analog-to-digital converter (ADC) and the overall readout and signal processing system used to generate the digital information equivalent to the single-pixel electrical response to each of the projected illumination patterns and generate the final 2D digital image. As it can be inferred from Eq. (1) [9], there is a trade-off to be considered here. In SPI, F_{min} depends directly on the amount of illumination patterns sensed. Thus, reducing the amount of projected illumination patterns, or reducing the minimum required processing time of the system to generate an image based on the single-pixel response to those illumination patterns, will both negatively influence the spatial resolution and the overall quality of the final generated 2D image. On the other hand, increasing the image spatial resolution and quality will diminish the video-stream frequency by reducing the amount of possible fps.

$$F_{\min} = F_{ADC} / F_{pattern} \tag{1}$$

The parameter F_{min} depends on the amount of real pixels (F) existent in the system of interest used to generate the final 2D image. Ideally, F and F_{min} should be the same -as in most image sensors-, yet in SPI systems F equals 1. Following Eq. (1), in order to obtain the minimum required frame-rate (of at least 33 fps), firstly we need to define the minimum required F_{min} that can be used by a drone navigation system to identify the objects around it, and secondly, determine the highest ADC rate possible (yielding the minimum necessary SNR related to the number of ADC output bits required), combined with the shortest T_{int} possible, imposed by the biggest distance targeted of the objects in the scene, their lowest reflection index, the highest possible irradiance of the LED array used, and finally the highest background illumination permitted. The ADC resolution, related to its SNR, is defined as the ratio between the photodetector output electrical signal -generated by the amount of light impinging its photoactive area that was firstly emitted by an array of LEDs and then reflected by the objects in the illuminated scene-, and the system noise floor. This noise floor is mainly determined by the background radiation of up to 100,000 lux if the system is to be used in outdoors that produces a photon shot noise following *Poisson* distribution, and adds to the system overall read noise.

It will be important to calculate the minimum number of patterns ($L_{pattern}$) that will be used to reconstruct the image. The number of patterns can be calculated using Eq. (2) [9], where CF is the compression factor of 3% for a grayscale image defined as CF = 8 / n_{Bpp}. Here, n_{Bpp} stands for the number of generated bits per pixel output. $M \times N$ corresponds to the size in terms of pixel columns and rows of the reconstructed image that should yield a maximum F_{min} possible if a maximum number of patterns ($L_{pattern}$) allowing for a video stream with 33 fps is set to 128. In the same tenor, the illumination pattern capture and processing time should not exceed 10 ms, condition met to keep the image generation time below 30 ms, i.e. $F_{pattern}$ = 30 kHz, and F_{ADC} = 100 kHz.

$$L_{pattern} = CF(M \times N) \tag{2}$$

3. OMP ALGORITHM VARIATIONS APPLIED TO SPI IMAGE RECONSTRUCTION

As explained above, single image processing time is a critical factor for generation of 2D digital images and video streams in SPI systems to be used e.g. during drone navigation. So, aiming at reducing the time required by the system for 2D image processing to an absolute minimum, the approach followed in the present work is drastically reducing the amount of illumination patterns and thus available reconstruction data, and then performing the actual reconstruction of the gathered data adapting the CS approach [9] in form of the OMP algorithm variations, namely: (1) the standard OMP algorithm [2, 3], (2) the algorithm based on the OMP using the Inverse Cholesky Factorization [6], and finally (3) using the Batch-OMP algorithm [7]. The latter was implemented using a parallel and reconfigurable architecture divided into three parallel kernels [4] in order to reduce the processing time and improve its efficiency. For the evaluation of the algorithm performance, we made a comparison in terms of the process execution time using respectively a CPU based hardware architecture and a GPU based one, maintaining PSNR in all cases above 20 dB (an absolute minimum for the application of choice).

3.1 The Standard Orthogonal Matching Pursuit (OMP) algorithm

For image processing applications, if the CS approach is pursued, at first the equation $y = \Phi x$ needs to be solved, where Φ is the *dictionary* matrix, y is the number of measurement output values obtained from the detector, and x is the final 2D image to be reconstructed. Following the OMP algorithm, the measurement signal y and the new measurement matrix Φ are initially considered as input. The principal idea behind the OMP method is finding the space solution to the linear system $y = \Phi x$, using a dictionary matrix function defined as $\Phi = (\Phi_{:1}\ \Phi_{:2}\ \dots\ \Phi_{:N})$, where $\Phi_{:i} \in \mathbb{C}^M$ denotes the i-th column of Φ, called the *atom*. The OMP algorithm seeks iteratively to select the best *atom* in the matrix able to reduce the residual r (the approximating error) to an acceptable minimum, or reach a previously defined accuracy, as expressed in Eqs. (3) and (4). Once the optimum *atom* is found, the Φ weights can be renewed and *least-square* approximations performed to recover the signal (or reconstruct the image in this case).

$$\hat{\Phi} = \arg\min_{\alpha} \|y - \Phi x\|_2^2 \qquad \|\phi\|_0 \le K \tag{3}$$

$$\hat{\Phi} = \arg\min_{\alpha} \|\Phi\|_0 \qquad \|y - \Phi x\|_2^2 \le \varepsilon \tag{4}$$

In Eqs. (3) and (4) $\boldsymbol{\Phi}$ represents the recovering dictionary for the $N \times K$ matrix, which can be defined as a random matrix [10], a chaos theory based random matrix following the $4z_k(1\text{-}z_k)$ relation, or can follow a predefined designated pattern to be used for image reconstruction, as for example proposed by Hadamard [10], where $H_n H_n^T = nI_n$, or using the Discrete Cosine Transformation (DCT) normally used in image processing [10], defined in Eq. (5).

$$\Phi_k(n) = \frac{\cos\left(\frac{\pi}{K}(k-1)(n-1)\right)}{\sqrt{\sum_{n=1}^{N}\cos^2\left(\frac{\pi}{K}(k-1)(n-1)\right)}}, \; n = 1,\ldots,N \; , \; k = 1,\ldots,K \tag{5}$$

To implement the standard OMP algorithm, the three main pursued processes that constitute this method were respectively fed into three separate kernel software units: i) the dot product of Φ^T (matrix size: M×N), ii) locating the maximum value within the $|<(\Phi)^{-1}\cdot x>|$ matrix for matrix size of (N×1), and iii) implementing the Least Square algorithm. It should be considered that Least Square is the most complex kernel based on the $x = ((\Phi^T\Phi)^{-1}\Phi^T)\cdot y$ relation, since it must perform multiplication, transposition and inversion operations, the sizes of which depend on the considered *k*-spaces and the number of iterations required [7, 9]. A detailed description of the standard OMP algorithm is presented as follows.

Algorithm 1: Standard Orthogonal Matching Pursuit (OMP)
OMP algorithm input data: Dictionary Φ, input signal y, target sparsity K
OMP algorithm output data: sparse representation x that fulfills the relation $y=\Phi x$

Detailed algorithm sequence:

1: set I = {0}, $r = x$, $\Phi = 0$
2: ***while*** *(stopping test false)* ***do***
3: $proj = \Phi^T r$ # Matching atoms with residual *r*
4: $k=arg\ max_k |proj|$ # Finding the new atom that best matches the residual *r*
5: $I = (I, k)$ # Support update
6: $x = ((\Phi^T\Phi)^{-1}\Phi^T)\, y$ # Calculate pseudoinverse least squares
7: $\bar{y} = \Phi x$ # Approximation update
8: $r = y - \bar{y}$ # Calculate the new residual *r*
9: $\|y - \Phi x\|_2 \leq \varepsilon$ # Calculate the target error ε
10: ***end while***

3.1.1 The OMP algorithm using the progressive Inverse Cholesky Factorization

The standard OMP algorithm is a process for matrix inversion that requires huge amounts of computational resources and thus causes high computational costs. So, we propose using instead the algorithm based on the OMP but using the progressive Inverse Cholesky Factorization [6] to reduce the processing time and computational resources involved in the matrix inversion. The major computational complexity of the OMP algorithm resides in calculating this mentioned matrix inversion (line 6 in the detailed OMP algorithm sequence). This problem can be partially solved by applying the so-called Cholesky Factorization [6]. For the implementation of the Cholesky Factorization, we must define the so-called *Gram* matrix as $G = \Phi^T\Phi$ that is symmetric and positive. The Gram-matrix can be broken down into two triangular matrices using the Cholesky decomposition defined as $A = LL^T$, where L is the triangular Cholesky factor. To solve this matrix, at first the system $LL^T x = \Phi^T \bar{y}$ is to be defined, for which $b = \Phi\bar{y}$ is proposed so that the system can be turned into a triangle system defined as $Lu = b$, where $L^T x = u$. To be able to finally calculate L, the matrix shown in Eq. (6) is used, where $w = L^{-1}\Phi^T$.

$$L_{new} = \begin{bmatrix} L & 0 \\ w^T & \sqrt{1 - w^T w} \end{bmatrix} \tag{6}$$

3.1.2 The Batch OMP algorithm

During the OMP algorithm calculation using the Inverse Cholesky Factorization, the operation related to the definition of the Gram-matrix, $G = \Phi^T\Phi$, significantly extends the processing time. In order to reduce the processing time, a pre-calculated Gram-matrix G with an initial projection i is used, where $p^0=\Phi^T y$ is defined, accompanied by a stopping criterion in which the normalized residual r is compared to a user pre-defined threshold value, that eliminates the need to sequentially calculate this residual. The Batch OMP algorithm, if applied to 2D image reconstruction, provides an additional improvement if compared to the other two analyzed algorithms. The latter mainly due to the fact that the number of threads pursued gets defined by the pre-defined number of columns in the 2D image data matrix which eliminates the need to update the residual r. This significantly reduces the number of operations. To improve the efficiency of the algorithm even further, programming on the parallel computing platform and programming model CUDA from NVIDIA [11] is proposed, with the aim of parallelizing the reconstruction operation.

Algorithm 2: Batch-OMP algorithm

Batch OMP algorithm input data: Dictionary Φ, input signal y, target sparsity K
Batch OMP algorithm output data: sparse representation x that fulfills the relation $y \approx \Phi x$

Detailed algorithm sequence:

1: set $I = \{0\}$, $L = [1]$, $p^0 = \Phi^T y$, $\varepsilon = y \cdot y^T$, $i = 1$, $G = \Phi^T\Phi$
2: $p = p^0$ # Initial projection
3: ***while*** $(\varepsilon_{i-1} > \varepsilon)$ **do**
4: $k = \arg\max_k |p|$ # Finding the new *atom* $\Phi_{:i}$
5: ***if*** $i>1$ ***then*** # Cholesky update
6: w = Solve for w $\{L_{i-1}w = G_{i-1,k}\}$
$$L_i = \begin{bmatrix} L_{i-1} & 0 \\ w^T & \sqrt{1 - w^T w} \end{bmatrix}$$
7: # Update of the Cholesky decomposition
8: ***end if***
9: $I = (I, k)$ # Support update
10: x_i = Solve for $c\{LL^T x_i = p^0\}$
11: $\beta = Gx_i$ # Matrix-sparse-vector product for each path
12: $p = p^0 - \beta$
13: $\delta^k = x^T\beta$ # Calculate error
14: $\varepsilon^k = \varepsilon^{k-1} - \delta^k + \delta^{k-1}$ # Calculate normal error ε
15: ***end while***

4. BATCH-OMP ALGORITHM IMPLEMENTATION USING A GPU PLATFORM

For the implementation of the Batch-OMP algorithm, the CUDA [11] toolkit CUBLAS based libraries in *Python* were used. The application developed is oriented towards SPI reconstruction, wherefore the measured data for the 2D image matrix obtained from the single-pixel (photodetector) output signals are made available. Based on the description of the Batch-OMP Algorithm 2 (see above), at first 4 kernels in CUDA were defined for the algorithm parallel implementation [9, 12, 13]:

i. The first kernel the input information was defined, the Gram-matrix ($G = \Phi^T\Phi$) generated (line 1 of the Algorithm 2: Batch-OMP algorithm), and the residual norm r calculated
ii. The second kernel was used to calculate the new atom $\Phi_{:i}$ (line 4 of the Algorithm 2: Batch-OMP algorithm)

iii. The third kernel was used to calculate the Cholesky decomposition (lines 6 and 7 of the Algorithm 2: Batch-OMP algorithm), where the matrix N×N was defined in order to calculate the matrix *L* (see Eq. (6)

iv. The fourth kernel was used to calculate the matrix space-vector product (line 11 of the Algorithm 2: Batch-OMP algorithm), and also to calculate the normal error e (line 14 of the Algorithm 2: Batch-OMP algorithm)

4.1 Standard OMP and Batch-OMP performance evaluation

To evaluate the SPI 2D image reconstruction performance using respectively the Standard-OMP algorithm on the one hand, and the Batch-OMP algorithm on the other, focusing on the processing time required for each, they were implemented on both: a CPU based platform and a GPU based one. The following starting conditions were defined for the latter (see Fig. 1): i) the sparsity values *k* has chosen were 16 and 64, ii) the size of the 2D reconstructed end images was of 64×64, 64×16, 128×16, and 256×16 virtual pixels, respectively, iii) the images to be reconstructed were at first deconstructed using a chaos random pattern [10], generating the 16 and 64 sparsity values, respectively, and afterwards reconstructed again using the both algorithms under test applied to the sparsity values mentioned. The hardware platforms used to perform the tests were the i5 CPU and the *Jetson Nano* GPU [8]. The Batch-OMP algorithm implemented on the GPU platform ran 2.7 times faster than when it was implemented using the CPU based platform, and even 4.5 times faster than the Standard-OMP algorithm implemented on the same two platforms.

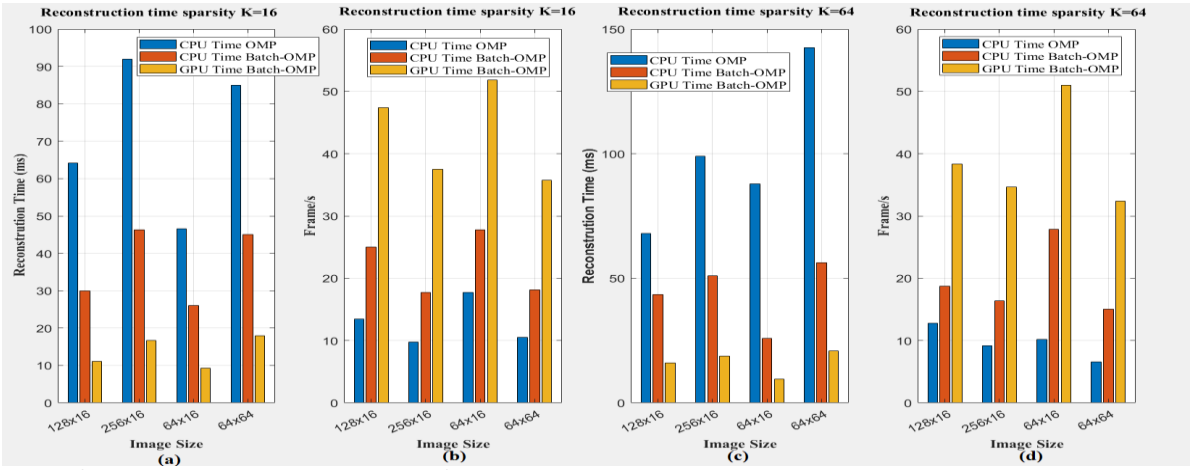

Figure 1. Performance evaluation results obtained for the Standard-OMP and Batch-OMP algorithms, respectively, running on the i5 CPU and the *Jetson Nano* GPU [10] platforms: (a) image reconstruction time required for the sparsity factor k = 16; b) frames per second (fps) obtained for the results shown in a); c) image reconstruction time required for the sparsity factor k = 64; d) frames per second (fps) obtained for the results shown in c). It can be observed that he GPU architecture allows for processing times shorter at least by factor 3 if compared to processing times required by the same algorithm implemented on CPU.

As it can be observed in Fig. 1, using the sparsity value k = 16, the standard-OMP algorithm running on the CPU platform required 64 ms to generate a 128×16 pixel image (see Fig. 1(a)), which was more than a double than the time (30 ms in fig. 1(a)) required by the Batch-OMP algorithm running on the same CPU. Interestingly, the Batch-OMP algorithm running on the GPU platform required only one third of the time (11 ms) if compared to the time it required for the same task when using the CPU platform. For all results the PSNR obtained was around 23 dB. The proportions hold more or less for the 256×16, 64×16, and 64×64 pixel images. For the sparsity value of 64, the Batch-OMP algorithm running on GPU required in average 20 ms to reconstruct the same images yielding PSNR = 23 dB. Figs. 1(b) and 1(d) show the same information presented in Figs. 1(a) and 1(c) but in terms of fps. An example of a reconstructed image with a size of 64×64 virtual pixels, reconstructed using the Batch-OMP algorithm running on the GPU platform can be observed in Fig. 2. The original image is shown in Fig. 2(a). As explained above, this image was deconstructed

using chaos theory and then reconstructed again. These reconstructed images shown for sparsity factors of k = 16 on Fig. 2(b) and k = 64 on Fig. 2(c) present a PSNR level of ~ 24 dB.

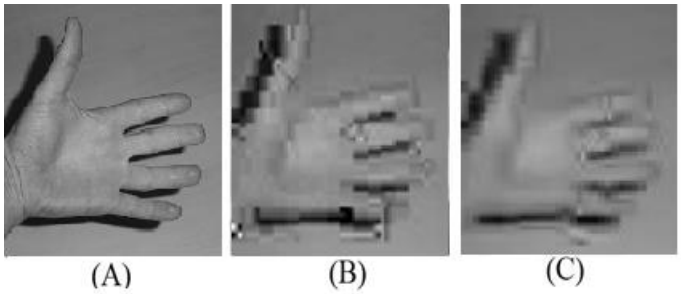

Figure 2. An example of a reconstructed image with a size of 64×64 virtual pixels, reconstructed using the Batch-OMP algorithm running on the GPU platform: (a) original image deconstructed down to a size of 64×64 pixels; (b) reconstructed image with a sparsity value of k = 16; and c) reconstructed image with a sparsity value of k = 64.

CONCLUSION

In this paper we analyzed the border conditions and system requirements for a single-pixel based imaging system developed to generate a video stream with at least 33 fps to be used for decision making in continuous time by an autonomous flying object (drone). The SPI approach requires an array of luminous spots, e.g. an array of LEDs, to emit different illumination patterns that will illuminate the scene to be imaged, be reflected by the different objects in that scene, and finally be detected by a single photodetector placed alongside the LED array. The spatial resolution (number of reconstructed virtual pixels) depends in this case on the amount of different illumination patterns generated. The amount of frames per second (fps) delivered by the generated video stream depends, on the other hand, on the time required for the emission of the mentioned illumination patterns, their amount, and the time required for their detection and processing. The latter imposes the necessity to reduce the data processing time to a minimum. In this regard, following the CS approach, we analyzed in detail and compared the standard Orthogonal Matching Pursuit (OMP) algorithm, the OMP-Cholesky algorithm variation, and finally the Batch-OMP algorithm, pointing out their advantages and disadvantages when implemented on both, CPU based and GPU based platforms, respectively, applying the parallel architecture approach. The evaluation showed that a combination of Batch-OMP and OMP-Cholesky algorithms offer the best results outperforming the standard OMP algorithm by more than a factor of 3 in what processing time is concerned if implemented on the same CPU based platform and even more than a factor of 5 if implemented on a GPU based platform using four parallel kernels. The study was carried out considering image sparsity values of 16 and 64, respectively, and a PSNR > 22 dB. Thus, it was shown that a video stream with 33 fps can be generated using the SPI approach with 2D image resolution of 64×64 virtual pixels and a PSNR < 20 dB if the combined Batch-OMP and OMP-Cholesky algorithm is implemented on a 4 kernel GPU platform. In this case, the system processing time for each generated 2D image is < 20 ms, and the time for the generation and detection of illumination patterns is of below 13 ms.

REFERENCES

[1] M. Rani, S. B. Dhok and R. B. Deshmukh "A Systematic Review of Compressive Sensing: Concepts, Implementations and Applications," in *IEEE Access*, vol. 6, pp. 4875-4894 (2018)

[2] Y. Lin, K. Han, W. Tan, B. Qian, W. Yanping, W. Hong ”Off-grid effect imaging methods based on improved OMP approach for DLLA 3D SAR”, Proc. *IET international Radar Conference*, doi: 10.1049/cp.2015.0978 (2015)

[3] J. Tropp and A. Gilbert “Signal recovery from random measurements via orthogonal matching pursuit“, *IEEE Transactions on Information Theory*, 53(12):4655-4666 (2007)

[4] A. Kulkarni and T. Mohsenin, “Accelerating compressive sensing reconstruction omp algorithm with cpu, gpu, fpga and domain specific many-core,” in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 970–973 (2015)

[5] M. P. Edgar, G. M. Gibson and M. J. Padgett “Principles and prospects for single-pixel imaging”, *Nature Photon* 13, 13–20, https://doi.org/10.1038/s41566-018-0300-7 (2019)

[6] H. Zhu, G. Yang and W. Chen "Efficient Implementations of Orthogonal Matching Pursuit Based on Inverse Cholesky Factorization", *IEEE 78th Vehicular Technology Conference (VTC Fall)*, Las Vegas, NV, 2013, pp. 1-5 (2013)

[7] R. Rubinstein, M. Zibulevsky and M. Elad "Efficient implementation of the k-svd algorithm using batch orthogonal matching pursuit". *Cs Technion*, 40(8):1–15 (2008)

[8] https://developer.nvidia.com/embedded/jetson-nano (last visited on March 30, 2020)

[9] J. Wang, M. Gupta, A. C. Sankaranarayanan “Lisens - a scalable architecture for video compressive sensing,” in *2015 IEEE International Conference on Computational Photography (ICCP)*, pp. 1–9 (2015)

[10] H. Gan, S. Xiao, T. Zhang, Z. Zhang, J. Li, Y. Gao "Chaotic Pattern Array for Single-Pixel Imaging ". *Electronics* 2019, 8(5), 536 (2019)

[11] https://developer.nvidia.com/cuda-zone (last visited on March 30, 2020)

[12] N. Myhrvold, “HAS Programmer’s Reference Manual Version 1.0.1”, http://www.hsafoundation.com/html/Content/PRM/Topics/07_Image/bits_per_pixel.htm (last seen on March 19, 2020)

[13] A. Borghi, J. Darbon, S. Peyronnet, T. F. Chan, S. Osher, "A simple compressive sensing algorithm for parallel many-core architectures", *J. Signal Process. Systems*, 71, 1–20 (2013)