Potential of an Embedded Hyperspectral Compressive Imaging System for Remote Sensing Applications

Olivier Lim, Stéphane Mancini, Mauro Dalla Mura

Performances of real-time compressive sensing hyperspectral imagers on an embedded system

Olivier Lim[1,2][0000−0001−7014−9255], Mauro Dalla Mura[2,3][0000−0002−9656−9087], and Stéphane Mancini[1]

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP, TIMA, Grenoble, France
{olivier.lim, stephane.mancini}@univ-grenoble-alpes.fr
[2] Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab, Grenoble, France
mauro.dalla-mura@gipsa-lab.grenoble-inp.fr
[3] Institut Universitaire de France (IUF), Paris, France

Abstract. In this work we study the ability of a Compressive Sensing (CS) based imager, the Disperser Coded Aperture Snapshot Spectral Imager (DD CASSI) to perform real-time hyperspectral imaging in real-time. CS theory allows to capture data below the Nyquist rate but needs a computationally expensive reconstruction phase to recover the data. The goal of this work is to study the limiting factors of the reconstruction process. Therefore, we propose to improve the performance estimations of a previous work that compared the caracteristics to off-the-self computing devices, i.e. a Graphics Processing Unit (GPU) and a Field-Programmable Gate Array (FPGA). Thanks to High Level Synthesis (HLS), we are able to get a timing analysis of the reconstruction process as if it was implemented in a circuit. Moreover, we will discuss the different bottlenecks of the implementation and suggest some leads to further improve the performances of the imaging system.

Keywords: Compressive sensing · Real-time · Hyperspectral imaging · Embedded systems.

1 Introduction

Hyperspectral imagering consists in measuring the spectral information of a scene in about a hundred of wavelengths. The representation of hyperspectral data is then called a "hyperspectral data cube". The rich spectral information has been used in various fields such as agriculture [1], geology [2] and medicine [3]. Conventional hyperspectral imagers operate by scanning scenes row-by-row, using diffraction gratings to separate spectral bands and capture band-specific information. This approach presents major issues. First, movement is required in order to capture the whole scene, which is time consuming, and for a given sensor size, this leads to a reduction of spatial resolution, since each additional spectral band further divides the sensor matrix. Plus, the whole data cube is captured which generates a huge amount of data that either need to be stored or transferred, which is problematic when considering embedded systems.

Hence, Compressed Sensing (CS) [4] theory has been used to overcome these issues. CS exploits the sparsity of the signals and allows to capture compressed data below the Nyquist rate and recover computationally the data in its entirety. For imagery purposes, it enables to use smaller sensor arrays, and thus requiring a lower memory footprint or data bandwidth, while keeping the same spatial and spectral resolution. However, CS requires a computationally expensive reconstruction phase to reconstruct the hyperspectral data cube from the raw compressed data. In literature, most CS related works focus on reconstruction quality [5,6]. A very few of them treat the reconstruction of the data on embedded systems, e.g. [7,8,9]. Lim et al. [10] assessed the characteristics of a CS-based hyperspectral imaging system, e.g. how much computing power is needed to achieve a certain level of performance. However, the characteristics are compared to off-the-self computing devices. Hence, the performances presented in their work are still theoretical.

Our goal is to study the performances of a CS imager for real-time applications, i.e. 10 Frames Per Second (FPS), or 100 ms for both the acquisition and reconstruction, on a embedded system, i.e. a Graphics Processing Unit (GPU) and a Field-Programmable Gate Array (FPGA). Therefore, in this paper we propose to the accuracy of performance estimates by conducting hardware simulations for a circuit design dedicated to the reconstruction phase. This approach provides realistic timing information regarding the reconstruction time. The design, implemented using High-Level Synthesis (HLS), incorporates hardware optimizations to leverage the parallelism inherent in the reconstruction algorithm. Furthermore, we discuss the limiting factors of the implementation and offer insights for further performance improvements.

In Section 2, we will first expose the method on which this work is based upon and more specifically the problem to solve. Then we will present an overview of the anticipated circuit and discuss essential considerations for selecting reconstruction parameters, along with their impact on reconstruction time. Section 3 will present the experiments conducted in this work and their results. Section 4 will discuss the aforementioned results, the limiting factors of the implementation and suggest some improvements in order to overcome the limitations. And finally Section 5 will conclude this work on real-time reconstructions based on the considered CS-based imaging system.

2 Methodology

2.1 Studied imaging system

As in Lim et al. [10], we consider the Disperser Coded Aperture Snapshot Spectral Imager (DD CASSI) [11] and the Conjugate Gradient for Normal Equation (CGNE) [12] for the reconstruction process. The DD CASSI integrates CS through a configurable coded spectral filter and light ray multiplexing. Improved spectral data can be obtained by conducting additional acquisitions with distinct codes. The CGNE algorithm, being iterative, relies on the number of iterations,

which directly impacts reconstruction time. Lim et al. demonstrated that enhancing both reconstruction quality and reducing the number of iterations is possible by increasing the number N of acquisitions. Reconstruction is performed row-by-row, instead of the whole scene at once, and then concatenated, minimizing CGNE variable memory usage and mitigating memory swapping concerns.
To reconstruct the hyperspectral cube, the problem to solve is:

$$\hat{\boldsymbol{o}} = \arg\min_{\boldsymbol{o}} \{\|\boldsymbol{d} - \boldsymbol{H}\boldsymbol{o}\|^2 + \mu_x \|\boldsymbol{D}_{\boldsymbol{x}}\boldsymbol{o}\|^2 + \mu_\lambda \|\boldsymbol{D}_{\boldsymbol{\lambda}}\boldsymbol{o}\|^2\} \tag{1}$$

where $\hat{\boldsymbol{o}}$ is the estimation of the observed scene $\boldsymbol{o}$, $\boldsymbol{d}$ is the raw data from the DD CASSI, $\boldsymbol{H}$ is the system matrix reproducing the DD CASSI's optical system, μs and $\boldsymbol{D}$ are respectively regularization coefficients and finite differences matrices used to enforce continuity in the spatial dimension x and the spectral dimension λ. And the CGNE is used to solve the linear system $\boldsymbol{A}\hat{\boldsymbol{o}} = \boldsymbol{b}$ with $\boldsymbol{A} = \boldsymbol{M}^\top \boldsymbol{M}$ where $\boldsymbol{M} = [\boldsymbol{H}, \sqrt{\mu_x}\boldsymbol{D}_{\boldsymbol{x}}, \sqrt{\mu_\lambda}\boldsymbol{D}_{\boldsymbol{\lambda}}]$ and $\boldsymbol{b} = \boldsymbol{H}^\top \boldsymbol{d}$. $\boldsymbol{d}$ is made of a set of N acquisitions with different codes. These codes and and the optical system model are incorporated in $\boldsymbol{H}$. We recall the CGNE algorithm in Algorithm 1.

Algorithm 1: CGNE algorithm. The convergence condition is satisfied when a specific precision threshold, denoted as *tol*, is achieved, and can be expressed as $\boldsymbol{r_i}^\top \boldsymbol{r_i} < tol$.

Compute $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$, $\boldsymbol{\rho}_0 = \boldsymbol{A}^\top \boldsymbol{r}_0$;
for *i = 0, 1, ... until convergence* **do**
  $\alpha_i = \boldsymbol{r}_i^\top \boldsymbol{r}_i / \boldsymbol{\rho}_i^\top \boldsymbol{\rho}_i$;
  $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha_i \boldsymbol{\rho}_i$;
  $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - \alpha_i \boldsymbol{A}\boldsymbol{\rho}_i$;
  $\beta_i = \boldsymbol{r}_{i+1}^\top \boldsymbol{r}_{i+1} / \boldsymbol{r}_i^\top \boldsymbol{r}_i$;
  $\boldsymbol{\rho}_{i+1} = \boldsymbol{A}^\top \boldsymbol{r}_{i+1} + \beta_i \boldsymbol{\rho}_i$
end

Lim et al. emphasized $\boldsymbol{H}$'s sparsity, suggesting a sparse matrix format to reduce computation and memory footprint. They found Compressed Sparse Row (CSR) format yielded the fastest reconstruction. Fixed-point representation is used to minimize memory usage, while maintaining accuracy and optimizing embedded system bandwidth. We recall in Algorithm 2 the matrix-vector product algorithm when using the CSR format. The matrix $\boldsymbol{A}$ is encoded into three arrays: *val*, *col* and *row*. For each row of $\boldsymbol{A}$, every non-zero entry is concatenated into *val*, the column indices of the non-zero entries are stored in *col* and *row* stores the index (in *row* and *col*) for the first and last entry.

2.2 Hardware implementation

Circuit outline Utilizing Algorithm 1, we can outline the circuit implementing the main loop of the CGNE, as depicted in Figure 1. This visualization aids in understanding the circuit's magnitude, particularly in terms of physical memory and mathematical operations. It also highlights the use of intermediary variables

Algorithm 2: CSR format matrix-vector product. Result is stored in $\boldsymbol{y}$ and $\boldsymbol{v}$ is the operand vector.

```
for int i=0; i<n; i++ do
    y[i] = 0.0;
    for int j=row[i]; j<row[i+1]; j++ do
        y[i] += val[j]*v[col[j]];
    end
end
```

(denoted as t) and emphasizes the algorithm's dependencies and concurrency. To enhance performance, the CGNE algorithm undergoes modifications. These include storing the results of both $\boldsymbol{r}_{i+1}^{\top}\boldsymbol{r}_{i+1}$ and $\boldsymbol{r}_{i}^{\top}\boldsymbol{r}_{i}$ in $rhoc$ and $rhoc$ respectively, preventing redundant computations. Additionally, $\boldsymbol{x}_{i+1}$, $\boldsymbol{r}_{i+1}$, and $\boldsymbol{\rho}_{i+1}$ are saved in the same registers as $\boldsymbol{x}_i$, $\boldsymbol{r}_i$, $\boldsymbol{\rho}_i$ since they are not required for the rest of the iteration. For readibility reasons, this is depicted with a dotted arrow on Figure 1 and also note that $\boldsymbol{A}$ and $\boldsymbol{A}^{\top}$ are stored in the CSR format, as described in Section 2.1.

Reconstruction parameters Given our constraint of real-time, we have to pick a number N of acquisitions. Although, increasing N improves both the reconstruction time and quality, there is a cap on the number of acquisitions that can be made due to the fact that increasing N also extends the acquisition time, consequently reducing the time available for the reconstruction process. Also, as N increases, the values in $\boldsymbol{A}$ and $\boldsymbol{b}$ also increase due to the inclusion of additional data. Consequently, when using fixed-point representation, increasing N leads to a larger memory footprint. This raises a concern: processing each scene row in parallel is feasible, but the level of parallelism is restricted by the maximum number of simultaneous CGNE instances that can be stored. This limit is determined by the algorithm's footprint and the memory capacity of the computing device, such as the FPGA Xilinx Ultrascale+ VU13P [13], which we consider here. It's important to note that while computing power does impose limitations on parallelism, the primary constraint is set by memory accesses, as elaborated further in Section 4.

Experimental procedure Initially, we replicate the experiments conducted by Lim et al., measuring the dynamic range of CGNE's variables to assess the memory footprint while using the fixed-point representation. It appears that $\boldsymbol{A}$ is sparse, comprising approximately 1% non-zero entries. This directly influences the number of operations in matrix-vector products, which is the predominant operation in the CGNE algorithm. Then, based on the defined reconstruction parameters, we use CatapultC, the HLS tool from Siemens, to generate a register-transfer level (RTL) design and perform timing analysis.

Thanks to the timing analysis provided by CatapultC, we measure t_{1iterN} the time required by 1 CGNE iteration for a given value of N.

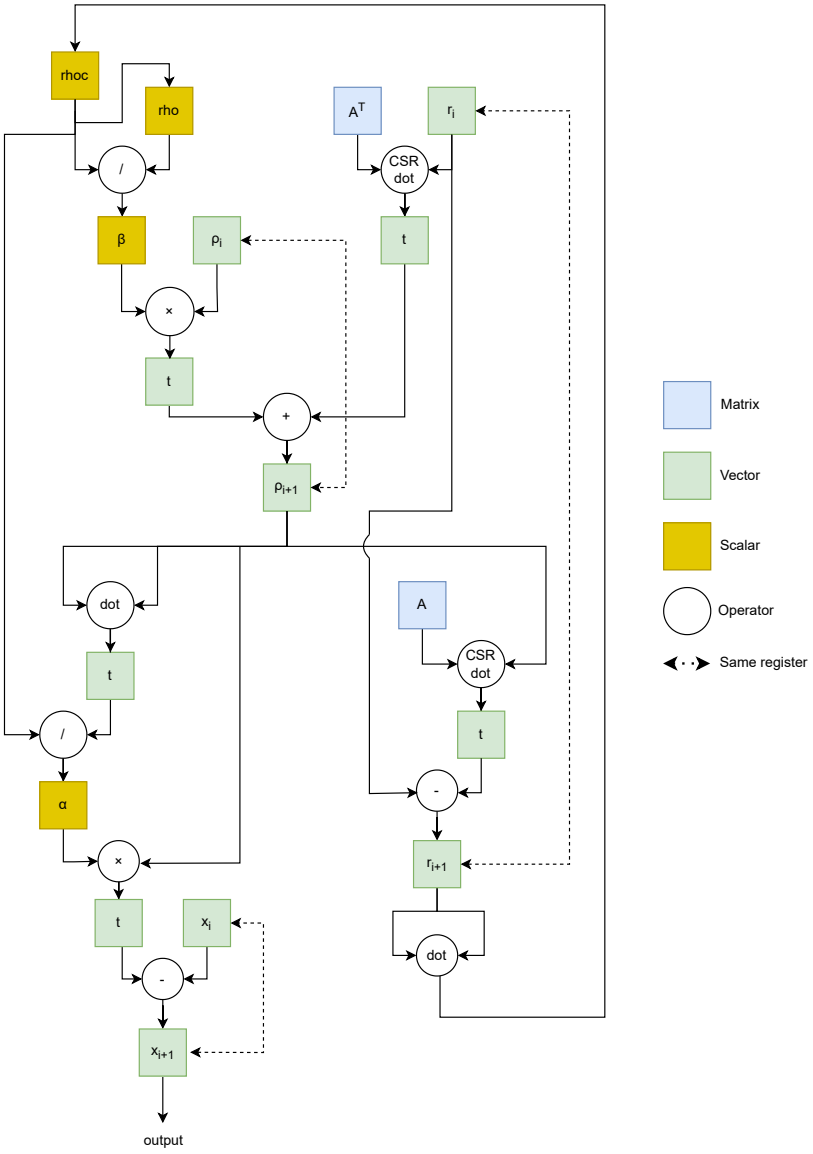


Fig. 1: Schematic of the row-by-row CGNE circuit design

Thus, we can determine the number of CGNE iterations, denoted as $ITERS_N$, that can be executed under real-time constraints. Each frame has $\frac{1}{target\_FPS}$ seconds available for both the acquisition and reconstruction phases. The acquisition step lasts $N \times camera\_rate$ seconds, leaving $\frac{1}{\text{target\_FPS}} - N \times \text{camera\_rate}$ seconds for the reconstruction process. Reconstruction must be performed for each row in the scene. However, parallelism enables multiple simultaneous re-

constructions. Hence, denoting R as the number of rows, the time available for the reconstruction process is divided by $\lceil R/parallelism\_level \rceil$. Finally dividing this duration by the time required by 1 CGNE iteration gives us $ITERS_N$. Thus,

$$\begin{aligned} ITERS_N &= \left\lfloor \frac{\left(\frac{1}{\text{target\_FPS}} - N \times \text{camera\_rate}\right)}{\frac{\lceil R/\text{parallelism\_level} \rceil}{t_{1iter}}} \right\rfloor \\ &= \left\lfloor \frac{1 - \text{target\_FPS} \times N \times \text{camera\_rate}}{\lceil R/\text{parallelism\_level} \rceil \times \text{target\_FPS} \times t_{1iterN}} \right\rfloor \end{aligned} \tag{2}$$

Then to ensure whether N is suitable or not, we measure the Peak Signal-to-Noise Ratio (PSNR) [14] of the reconstructions and set an arbitrary threshold of 30 dB.

3 Experiments

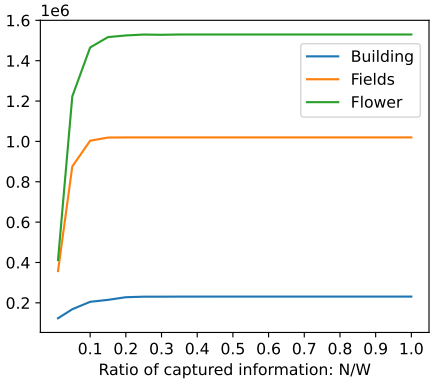
(a) *Building* (200 × 200 × 33) (b) *Fields* (100 × 100 × 100) (c) *Flower* (150 × 150 × 100)

Fig. 2: Scenes used in the simulations with their dimensions

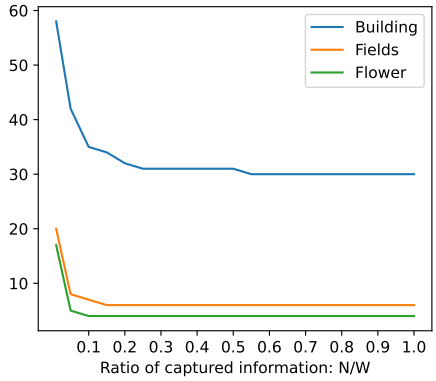
To encompass a wide range of applications, our simulations incorporate diverse subscenes sourced from different datasets and fields. "Building", a 200 × 200 × 33 scene from [15], "Fields", a scene 100 × 100 × 100 from [16] and "Flower", a 150 × 150 × 100 scene from [17], where $R \times C \times W$ denote respectively the number of rows, columns and wavelengths. The scenes are depicted in Figure 2.

We denote the Ratio of Captured Information (RCI) as $\frac{N}{W}$. As described in Section 2.2 and to determine $ITERS_N$, for each scene we study the number of non-zero entries in $\boldsymbol{A}$, the "parallelism level", that is the number of CGNE instances that can be run simultaneously, and the time required for a CGNE iteration. These elements are presented in Figure 3. By applying Equation (2), we evaluate $ITERS_N$, see Table 1. Please note that we consider a camera_rate of 1 000 FPS, that is one acquisition requires 1 ms. Finally, the quality reconstruction is measured.

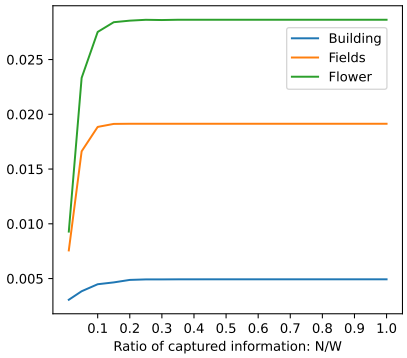
It is observed that the number of non-zero entries reaches a plateau due to the specific pattern followed by these entries in $\boldsymbol{M}$. With the increase in N, more random codes are included in $\boldsymbol{H}$, causing $\boldsymbol{A}$ to be filled with additional



(a) Number of non-zeros entries in $\boldsymbol{A}$



(b) Parallelism level



(c) Time for 1 CGNE iteration in seconds

Fig. 3: Criteria to determine $ITERS_N$

entries and eventually plateauing the amount of its non-zero entries. The parallelism level also follows a hyperbola. This can be explained as N increases, the memory footprint expands. However, the use of two's complement in fixed-point representation implies that each additional bit corresponds to a wider range of values. Regarding the time for 1 CGNE iteration, we can notice that it follows the same curve as the number of non-zero entries in $\boldsymbol{A}$. This is also expected since the matrix-vector products perform the multiplication with only the non-zero entries, thanks to the sparse matrix format. Additionally, it's essential to recognize that the presented times result from timing analysis conducted on a 45 nm circuit running at 115 MHz, as they are the technology node provided by CatapultC and the maximum achievable frequency while meeting timing constraints.

Regarding $ITERS_N$, it is worth noting that when RCI is greater than 0.01, executing a single iteration lasts at least 16.6 and 23.4 ms for *Fields* and *Flower*, respectively, rending the reconstruction unfeasible at 10 FPS. Compared to *Building*, this can be attributed to their resolutions. Because of this, the number of non-zero entries in $\boldsymbol{A}$ is also higher which expands the computation time. Moreover, since W is higher for *Fields* and *Flower*, for a fixed RCI, they require more acquisitions which also reduce the time available for the reconstruction process. In the end, for a target_FPS of 10, $ITERS_N$ for *Building* are presented in Table 1. At $N = 1$, we can execute 2 iterations for *Fields* and 1 for *Flower*.

Finally, for *Building*, the reconstruction quality ranges starts from 19.94 dB and reaches its peak at 25.67 dB for RCI = 0.9 or $N = 29$. The reconstruction quality for *Fields* and *Flower* reaches respectively 15.35 dB and 7.50 dB at RCI = 0.01 or $N = 1$.

As the obtained quality is below the desired threshold, we investigate the number of iterations required to reach a minimum quality of 30 dB, as depicted in Figure 4. We then examine the reconstruction rates at which this quality level can be reached. Our findings indicate that only the *Building* scene can attain a PSNR of 30 dB with a reconstruction rate of 1 FPS for $N = [23, 33]$.

Regarding the observations on Figure 4, it is noted that the number of iterations needed to reach a PSNR of 30 dB follows a hyperbolic trend. This pattern is expected, given that the relative information contributed by additional acquisitions decreases as N increases.

Table 1: Number of maximum iterations at 10 FPS

RCI	0.01	0.05	0.10 – 0.15	0.20 – 0.90	0.95–1.00
Building	8	5	3	3	1

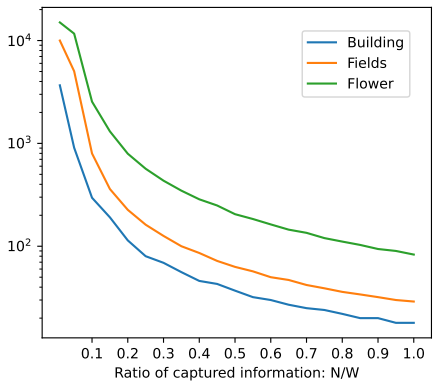


Fig. 4: Number of iterations to reach 30 dB

4 Discussion

When considering the analysis described by Lim et al., we estimate a reconstruction rate of 381, 164 and 107 FPS respectively for *Building*, *Fields*, and *Flower*, respectively. However, when we consider the reconstruction time given by the timing analysis made by CatapultC, the resulting reconstruction rate is significantly lower than anticipated, i.e 1 FPS for *Building*. The reason for this disparity is that Lim et al. do not consider the memory accesses during the computation, especially during the matrix-vector product, see Algorithm 2. They only consider the total amount of computations which assumes that every computation can be done at once.

It is important to highlight that the number of CGNE iterations is influenced by several parameters, including the dimensions of the scene, the number N of acquisitions, the targeted reconstruction quality, among others. Achieving a balance among these parameters is crucial. For instance, according to our method, a $20 \times 20 \times 33$ scene is capable of achieving real-time performance at 30 FPS for $N = [11, 33]$.

4.1 Perspectives

Computation time In order to speed up the the matrix-vector product, it would require to split the values of $\boldsymbol{A}$ and $\boldsymbol{A}^\top$ into several memory banks and perform parallel computations, see Figure 5 as example. As a result, it becomes possible to execute the matrix-vector product in a piecewise fashion, enabling parallel computations. Given that we have already implemented the CSR matrix-vector product, the required adjustment lies in adapting the memory organization of the circuit, as depicted in Figure 1. This requires partitioning the variables and applying the CSR matrix-vector product to the partitions of $\boldsymbol{A}$, $\boldsymbol{A}^\top$, $\boldsymbol{r}$, and $\boldsymbol{\rho}$, followed by concatenating the outcomes of the piecewise products. However, the introduction of intermediary variables becomes necessary as a consequence of storing the results from the split computations. Consequently, this leads to a minor increase in the memory footprint. Nevertheless, the positive trade-off is that the time required for the matrix-vector product is significantly reduced, as it gets divided by the number of partitions, namely 3 in Figure 5. This substantial improvement in computation time is particularly valuable, considering that it is the primary limiting factor in this context.

In this study, we have emphasized the impact of the number of non-zero entries in $\boldsymbol{A}$ and $\boldsymbol{A}^\top$ on the reconstruction time. As illustrated in Lim et al.'s work, the number of non-zero entries depends on C and W. Given that W has a quadratic influence on the number of non-zero entries, it may be worthwhile to explore methods to perform reconstructions with a lower W. For instance, it can be done either by splitting the spectral dimension or by performing a first reconstruction with a lower spectral resolution to serve as a starting point for a second reconstruction with the original spectral resolution, thereby potentially accelerating the overall process.

Moreover, to mitigate the impact of acquisition and increase the time available for reconstruction, we could consider scheduling acquisitions for the next frame to occur during the reconstruction process.

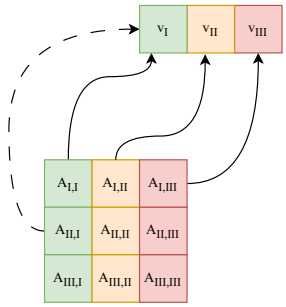


Fig. 5: Split matrix-vector product

Memory footprint Additionally, we came up with three ways to reduce the memory footprint.

Sparse matrix formats add an information overhead in order to store the column index for their entries, e.g. the array *col* for the CSR format. Here, $\boldsymbol{A}$ exhibits a pattern of diagonals along which the entries are located. It would be valuable to explore methods to exploit this pattern to avoid storing the column index for every entry, thereby reducing the overall memory footprint. One approach might involve devising a formula to encode the column index instead of storing it, or alternatively, transitioning to the Diagonal matrix format. However, it's worth noting that Lim et al. demonstrated that such a transition leads to slower reconstruction times.

Since most of the memory footprint is allocated to store $\boldsymbol{A}$ and $\boldsymbol{A}^\top$, adopting a sparse matrix format capable of efficiently conducting matrix-vector products with its transpose would significantly enhance memory efficiency. The Coordinate format could be used, but again Lim et al. showed that it offers slower reconstruction time.

Throughout this work we observed minimal variation in the values of $\boldsymbol{A}$'s entries. For instance, in the case of *Building* with $N = 1$, $\boldsymbol{A}$ encompassed only 11 distinct values among approximately 127,000 non-zero entries. Consequently, employing a data structure akin to a dictionary could considerably reduce the memory footprint of $\boldsymbol{A}$. However, it is reasonable to anticipate a slight slowdown in the matrix-vector product due to the addition of a decoding stage.

4.2 Limitations

The presented performances in this study could potentially be underestimated for two reasons. Firstly, to guarantee stable circuit signals, CatapultC incorporates a margin for hold times, reducing the maximum achieviable frequency.

Secondly, the HLS tool performs the synthesis step in the circuit design workflow. The subsequent place and route step is crucial for optimizing the placement of circuit components and minimizing the path between them, potentially enhancing overall performance.

Conclusion:

CS is an emerging technique that allows to capture a signal under the Nyquist rate but requires huge computations to recover the original signal. In this context, we studied the performances of employing CS to perform real-time hyperspectral imagery on embedded systems. For this purpose, we relied on the work of Lim et al. [10]. We propose to provide more realistic estimations using HLS for conducting timing analysis of a circuit performing the reconstruction for the same imaging system. Additionally, we outline crucial considerations for hardware implementation and identify the bottlenecks in the imaging system, primarily related to matrix-product operations. These bottlenecks might find resolution through the adaptation or substitution of the CSR sparse matrix format, even though it currently provides the most efficient reconstruction time among conventional sparse matrix formats.

In terms of performance, it can be concluded that the proposed naive implementation struggles to achieve satisfactory reconstructions with a satisfactory rate, e.g. 1 FPS for a $200 \times 200 \times 33$ scene. However, it's worth noting that this reconstruction rate is reported for a 45 nm circuit, a technology node that is relatively large by today's standards and could operate at 115 MHz without violating timing constraints. In comparison to state-of-the-art solutions, the imaging system could potentially outperform them. For example, [7] achieve 10 FPS for a $230 \times 198 \times 28$ scene but use a Nvidia Titan-X which operates at approximately 1 500 MHz. With the the same frequency, our naive implementation can reach 12 FPS for a $200 \times 200 \times 33$ scene.

To conclude, future works should focus on implementing the improvements proposed in this work. This includes the exploration of a partitioned matrix-vector product and the adoption of a more suitable sparse matrix format that maximizes the utilization of entry patterns in $\boldsymbol{A}$ without compromising existing performance. Additionally, investigating alternative algorithms to the CGNE or considering other imagers could be beneficial. Furthermore, now that the primary bottlenecks are identified, it becomes pertinent to explore new theoretical models for estimating reconstruction time. The critical challenge lies in accurately identifying the hardware and algorithm optimizations such as loop merging.

References

1. Bing Lu, Phuong D. Dao, Jiangui Liu, Yuhong He, and Jiali Shang. Recent advances of hyperspectral imaging technology and applications in agriculture. *Remote Sensing*, 12(16), 2020.

2. Sima Peyghambari and Yun Zhang. Hyperspectral remote sensing in lithological mapping, mineral exploration, and environmental geology: an updated review. *Journal of Applied Remote Sensing*, 15(3):031501, 2021.
3. Hamed Akbari, Kuniaki Uto, Yukio Kosugi, Kazuyuki Kojima, and Naofumi Tanaka. Cancer detection using infrared hyperspectral imaging. *Cancer science*, 102(4):852–857, 2011.
4. Emmanuel Candès and Justin Romberg. Sparsity and incoherence in compressive sampling. *Inverse Problems*, 23(3):969–985, Apr 2007.
5. Grigorios Tsagkatakis and Panagiotis Tsakalides. Compressed hyperspectral sensing. In *Image Sensors and Imaging Systems 2015*, volume 9403, pages 40–48. SPIE, 2015.
6. Ajit Rajwade, David Kittle, Tsung-Han Tsai, David Brady, and Lawrence Carin. Coded hyperspectral imaging and blind compressive sensing. *SIAM Journal on Imaging Sciences*, 6(2):782–812, 2013.
7. Shipeng Zhang, Hua Huang, and Ying Fu. Fast parallel implementation of dual-camera compressive hyperspectral imaging system. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(11):3404–3414, 2019.
8. Daniel Gedalin, Yaniv Oiknine, and Adrian Stern. Deepcubenet: reconstruction of spectrally compressive sensed hyperspectral images with deep neural networks. *Opt. Express*, 27(24):35811–35822, Nov 2019.
9. Lizhi Wang, Chen Sun, Ying Fu, Min H. Kim, and Hua Huang. Hyperspectral image reconstruction using a deep spatial-spectral prior. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8024–8033, 2019.
10. Olivier Lim, Stéphane Mancini, and Mauro Dalla Mura. Feasibility of a real-time embedded hyperspectral compressive sensing imaging system. *Sensors*, 22(24), 2022.
11. M. E. Gehm, R. John, D. J. Brady, R. M. Willett, and T. J. Schulz. Single-shot compressive spectral imaging with a dual-disperser architecture. *Opt. Express*, 15(21):14013–14027, Oct 2007.
12. M. Hanke. *Conjugate Gradient Type Methods for Ill-Posed Problems*. CRC Press, 2017.
13. Xilinx. Fpgas & 3d ics. `https://www.xilinx.com/products/silicon-devices/fpga.html`.
14. Kim-Han Thung and Paramesran Raveendran. A survey of image quality measures. In *2009 International Conference for Technical Postgraduates (TECHPOS)*, pages 1–4, 2009.
15. David H. Foster, Kinjiro Amano, Sérgio M. C. Nascimento, and Michael J. Foster. Frequency of metamerism in natural scenes. *J. Opt. Soc. Am. A*, 23(10):2359–2372, Oct 2006.
16. NASA. Aviris - airborne visible / infrared imaging spectrometer. `https://aviris.jpl.nasa.gov/index.html`.
17. Haris Ahmad Khan, Sofiane Mihoubi, Benjamin Mathon, Jean-Baptiste Thomas, and Jon Yngve Hardeberg. Hytexila: High resolution visible and near infrared hyperspectral texture images. *Sensors*, 18(7), 2018.