Energy Efficient Approximate 3D Image Reconstruction

YUN WU, (Member, IEEE), ANDREAS ÀßMANN, (Member, IEEE), BRIAN D. STEWART, AND ANDREW M. WALLACE

Yun Wu and Andrew M. Wallace are with the School of Engineering and Physical Sciences (EPS), Heriot-Watt University, EH14 4AS Edinburgh, U.K.
Andreas Àßmann and Brian D. Stewart are with the STMicroelectronics R&D Ltd., EH3 5DA Edinburgh, U.K.
CORRESPONDING AUTHOR: YUN WU (y.wu@hw.ac.uk)

ABSTRACT We demonstrate an efficient and accelerated parallel, sparse depth reconstruction framework using compressed sensing (compressed sensing (CS)) and approximate computing. Employing data parallelism for rapid image formation, the depth image is reconstructed from sparsely sampled scenes using convex optimization. Coupled with faster imaging, this sparse sampling reduces significantly the projected laser power in active systems such as light detection and ranging (LiDAR) to allow eye safe operation at longer range. We also demonstrate how reduced precision is leveraged to reduce the number of logic units in field-programmable gate array (FPGA) implementations for such sparse imaging systems. It enables significant reduction in logic units, memory requirements and power consumption by over 80% with minimal impact on the quality of reconstruction. To further accelerate processing, pre-computed, important components of the lower-upper (LU) decomposition and other linear algebraic computations are used to solve the convex optimization problems. Our methodology is demonstrated by the application of the alternating direction method of multipliers (ADMM) and proximal gradient descent (PGD) algorithms. For comparison, a fully discrete least square reconstruction method (dSparse) is also presented. This demonstrates the feasibility of novel, high resolution, low power and high frame rate LiDAR depth imagers based on sparse illumination for use in applications where resources are strictly limited.

INDEX TERMS Approximate computing, parallel computing, FPGA, LiDAR, depth reconstruction, compressed sensing, convex optimisation

I. INTRODUCTION

The recovery of dense 3D images from LiDAR sensors is a challenging task, especially at high resolutions and frame rates, and with constraints on eye safety that limit the signal-to-noise ratio (SNR) and operating range as these are highly dependent on the emitted power. Further, a large amount of raw data is produced by the sensor which requires significant storage and processing power when employing expensive computational methods. This prohibits its usage in resource constrained systems when limited power or physical space are available, for example in battery powered mobile devices, simultaneous location and mapping (SLAM) using a drone [1], and augmented reality [2].

Depth reconstruction, the key to 3D information recovery, is often ill-posed and relies on the solution of inverse problems [3]. To enable real time perception for higher resolution in all three dimensions, significant computing power and memory is required to recover the desired information at scale [4]. To maintain high frame rates, massively parallel compressed sensing (CS) enables the capture and recovery of 3D images quickly enough for dynamic scenarios by reducing the number of measurements, and thus memory, to maximize use of the limited data bandwidth. Sparse illumination further enables eye-safe long range low power LiDAR with block based parallelism [5]. For resource efficient implementation, each processing unit needs to have as few logic and memory gates as possible to minimize both power consumption and area.

Approximate computing (AC) is a promising approach in resource constrained systems, enabling energy efficiency [6]. This has been widely adopted in many areas such as signal
processing [7], robotics [8], and machine learning [9]. Reduced precision (RP) computation, a common AC technique, represents the arithmetic data with less bits throughout the computational stack [10] to reduce resources, specifically, memory and logic units, and hence reduces the energy consumption. When dealing with computational units in proximity to photon detectors it is also crucial to decrease the processing chip power and thus reduce the direct heat to all photon sensitive elements to minimize device noise [11].

Using full precision, researchers have explored the use of more efficient algorithms for compressive depth reconstruction using convex optimisation with algorithms such as the parallel alternating direction method of multipliers (ADMM) [5] and accelerated proximal gradient descent (PGD) [12] for depth reconstruction. Several authors have investigated the use of RP for convex optimization. For example, Rateb et al. [13] employed a fixed point precision quadratic programming (QP) $\ell_1$-Solver, and a half floating point ADMM was investigated by Wills et al. [14]. An approximate stochastic gradient descent (SGD) method was introduced [15] with fewer quantisation steps; this allowed a faster implementation with a slight reduction in performance. Although efficient hardware implementations of compressed sensing algorithms have been investigated, it is unclear how approximate depth reconstruction can be performed with massively parallel CS solutions using ADMM and PGD.

In this paper, we report the results of a study on approximate 3D image reconstruction using $\ell_1$ minimization based on RP for small scale least absolute shrinkage and selection operator (lasso) problems with the optimized lean ADMM and PGD algorithms on a FPGA platform. Both optimizers are suitable for compressed sensing of images, in our case depth images using time-of-flight sensing [5], [16]. A discrete solver for sparsely illuminated depth sampling, $\dsparse$, is also investigated to assess the impact of additional measurements on resource usage using RP when compared to our CS solutions. By analyzing the effects of the different RP decimal formats on logic and memory resources with respect to their qualitative and quantitative performance, the savings in power and data compression throughout the sensing and computing stack are evaluated.

**II. BACKGROUND**

**A. PARALLEL DEPTH RECONSTRUCTION**

In a time-correlated single photon LiDAR imaging system, each pixel can capture a set of returned photon events which are often stored in a histogram, $h \in \mathbb{N}^l$ of the type shown in Figure 1, where $l$ is the number of bins or channels [18]–[20]. Therefore, for a vectorised array of $N$ pixels, the raw measurements are $H \in \mathbb{R}^{N \times l}$, where $l$ ranges from tens to thousands of bins depending on the system’s timing and thus depth resolution. As larger and larger solid-state photon detectors are developed [11], [21], [22] and potential LiDAR resolution increases, the storage of raw measurements is challenging.

To address this, we presented earlier a depth reconstruction framework [5] using compressive sensing principles [23]–[25]. The original algorithm [16] was made parallel such that the pixel matrix of histograms was divided into smaller blocks, as illustrated in Figure 2, to enable fast processing times. The use of sparse structured illumination also reduces the total emitted power. This is beneficial as it makes LiDAR systems eye safe, and in the context of photon sensitive measurements, can lower the sensing device temperature. Noise is also reduced as temperature decreases [11].

As shown, the entire array of $N$ sensing pixels is divided into blocks of size $n < N$, which independently sample the scene in a compressed strategy to form $p$ measurements using predefined patterns stored in a sensing matrix $A \in \mathbb{R}^{p \times n}$. The depth image reconstruction problem is formulated as a set of parallel, problems on each smaller pixel block. By exploiting sparsity in the signal, we can reconstruct two
proxy quantities to recover depth. These are the so-called time-of-flight (ToF)-sum $x_Q \in \mathbb{R}^n$ and intensity (photon count) $x_I \in \mathbb{R}^n$ [5], [16]. By adopting our blocking scheme, information is recovered from smaller independent measured feature vectors, $y_Q, y_I \in \mathbb{R}^p$ (shown as Q and I in Figure 2) using the compressed sensing matrix, $A$, such that $y_Q = A \cdot x_Q$ and $y_I = A \cdot x_I$ where $p < n$ [5], [16]. Hence, $y_Q$ and $y_I$ are derived from sampled photon count histogram data such as that shown in Figure 1, measured by the LiDAR system.

This leads to parallel reconstruction as a pair of optimization problems for each block as
\begin{equation}
\begin{aligned}
\text{arg min}_{x_Q} \left( \frac{1}{2} \| y_Q - A \cdot x_Q \|_2^2 + \lambda \| F \cdot x_Q \|_1 \right), \\
\text{arg min}_{x_I} \left( \frac{1}{2} \| y_I - A \cdot x_I \|_2^2 + \lambda \| F \cdot x_I \|_1 \right),
\end{aligned}
\end{equation}
where $A \in \mathbb{R}^{p \times n}$ is the compressed sensing matrix, $\| \cdot \|_2$ is the Euclidean norm, $\lambda > 0$ is a regularization parameter to $\| \cdot \|_1 = \sum_i | \cdot |$ being the pseudo $\ell_1$-norm and $F$ is a linear transformation. This formulation is known as the lasso [27] algorithm, which has been widely adopted in image reconstruction and de-noising [24], [27]–[29].

We assume sparsity in the discrete cosine transform (DCT) domain i.e $F = \text{DCT}(\cdot)$. Due to the small problem size of each parallel reconstruction, the computational cost per block is greatly reduced. Assuming a single primary reflective surface per pixel [5], [16], this enables good estimation of depth by computing $x_D = x_Q / x_I$ for each block. Our parallel CS depth reconstruction scheme increases frame rate and reduces pixel storage requirements. To further improve efficiency and reduce resource consumption, we now show how we can reduce the precision of the arithmetic operations when using $\ell$ADMM, $\ell$PGD and dSparse.

B. ARITHMETIC PRECISION AND APPROXIMATION

Reduced precision (RP) arithmetic is an approximation technique in which the bit width in a data structure is shortened, enabling low cost and energy-efficient hardware implementations for computation and memory logic [10]. By exploiting different arithmetic data types, we can scale the area and power relative to the bit width and increase the density of operational units within the same area and power allowance.

Here, we use three main categories of representation shown in Figure 3: floating point [30], [31], Q-format fixed point [32], and universal numbers (unum) [33].

Figure 3(a) shows the binary structure of the IEEE 754 floating point arithmetic standard, representing a number by three elements: a sign (1 bit), an exponent ($e$ bits), and a mantissa ($m$ bits). That is,
\[ \text{sign} \times \text{mantissa} \times 2^{\text{exponent}-127}. \] (2)

Figure 3(b) shows the binary structure of the Q-format fixed point, using instead a fixed number of bits to represent the integer and the fractional parts of a number. Specifically, a number is represented by its sign (1 bit), its integer part ($i$ bits), and its fractional part ($f$ bits):
\[ \text{sign} \times (2^{\text{integer}} + 2^{-\text{fraction}}). \] (3)

Figure 3(c) shows the binary structure of unum Type III - Posit as an alternative to IEEE 754. This represents numbers by their sign (1 bits), regime ($p$ bits), exponent ($e$ bits), and fraction ($c$ bits):
\[ \text{sign} \times (2^p)^k \times 2^{\text{exponent}} \times \left(1 + \frac{\text{fraction}}{2^c}\right). \] (4)

where
\[ k = \begin{cases} -p & \text{first bit of regime is 0,} \\ p - 1 & \text{first bit of regime is 1.} \end{cases} \] (5)

Each of these arithmetic types has a different dynamic range of number representation. The floating point and unum representations use less bits but the fixed point representation allows simpler binary operations. This affects not only the performance of an algorithm by limiting the type of operations that can be performed, but also their embedded implementation.

In iterative algorithms any approximation made by reducing the precision will cascade at each loop iteration and errors will accumulate. If the precision is not sufficient, the accumulated error will be unacceptable. For depth imaging with sparse signal assumptions the iterative algorithm is performed twice, concurrently, for $x_Q$ and $x_I$, which are combined in the final recovery step. Thus, the effect of accumulated approximation effects is magnified. For example, with very low precision, significant signal components could be forced to zero too early and thus degrade consecutive computations.

We investigate the effects of approximation within the iterative approach of $\ell_1$ optimization using the alternating direction method of multipliers (ADMM) and proximal gradient descent (PGD) [29] algorithms, and within a non-iterative approach, the pseudo-inverse least-squares (dSparse) algorithm. This is applied to the parallel depth reconstruction problem, reducing resource usage on reconfigurable platforms, such as an FPGA, minimising logic units and memory use and increasing frame rate. We aim to find optimal number representation and bit width for both the iterative and non-iterative frameworks by evaluating the effects of reduced precision on logic reduction, data compression, power, time and reconstruction quality.
III. ALGORITHMS FOR DEPTH RECONSTRUCTION

A. lean ADMM

The standard ADMM implementation, shown in Algorithm 1 [29], requires $O(n)$ flops using a soft-thresholding operator $S$ and a threshold $\lambda_{ADMM}/\rho$. The large matrix inversion term (Step 1 in Algorithm 1) remains constant throughout and can be pre-computed and stored in a look-up table. Further, one can empirically preset values for all the regularization weights.

Algorithm 1. General ADMM lasso [29]

\[
\text{minimize } \frac{1}{2} \|y - Ax\|^2_2 + \lambda_{ADMM} \|z\|_1 \\
\text{subject to } Fx - z = 0
\]

Iterate:
1. $x^{k+1} = (A^T \cdot A + \rho \cdot F^T \cdot F)^{-1} \cdot (A^T \cdot y + \rho \cdot F^T \cdot (z^k - u^k))$
2. $\hat{z}^{k+1} = S_{\text{ADMM}/\rho}(F \cdot x^{k+1} + u^k)$
3. $u^{k+1} = u^k + F \cdot x^{k+1} - \hat{z}^{k+1}$

By fixing the number of iterations, rather than check convergence at each iteration, the computation can also be reduced, depending on the complexity of the convergence check compared to an iteration. Moreover, using a fixed number of iterations has the considerable advantage of making our implementation on the FPGA deterministic and hence quantifiable in terms of resource usage.

Algorithm 2. lean ADMM lasso for $p < n$

Input: $A, A^T, y, L^{-1}, U^{-1}, \lambda_{ADMM}, y$

Output: $x$

Initialization: $\text{const} \\{\alpha, \rho, \kappa = \lambda_{ADMM}/\rho\}, \text{zeros} \\{z, q, u\}$
1. for $k = 0$ to $k_{\max}$ do
2. $\hat{q}^{k+1} = A^T \cdot y + \rho \cdot (z^k - u^k)$
3. $x^{k+1} = \hat{q}^{k+1} / (\rho - \alpha) \cdot \alpha \cdot A^T \cdot \left( (U^{-1} \cdot (L^{-1} \cdot (A \cdot q^{k+1}))) \right)$
4. $\hat{\kappa}^{k+1} = -\alpha \cdot x^{k+1} + (1 - \alpha) \cdot \kappa$
5. $x^{k+1} = x^{k+1} - \kappa$
6. $\hat{z}^{k+1} = \max(0, u^{k+1} - \kappa); \quad z^{k+1} = \max(0, -x^{k+1} - \kappa)$
7. $\hat{z}^{k+1} = \max(0, u^{k+1} - \kappa); \quad z^{k+1} = \max(0, -x^{k+1} - \kappa)$
8. $u^{k+1} = u^k + (z^{k+1} - \hat{z}^{k+1})$
9. end for
10. $x = \hat{z}^{\text{max}}$
11. return $x$

We have modified the standard ADMM implementation [29] to give Algorithm 2 showing what pre-computations are feasible for the compressive sensing problem when we have full control over the sensing matrix, $A$. After careful analysis of the quadratic programming aspects of the ADMM implementation, inversion of $A$ is often non-trivial. However, LU decomposition is used and these matrices are inverted to compute the $x$-update. For a constant matrix $A$, these inversions can be pre-computed, $\{L^{-1}, U^{-1}\} \in \mathbb{R}^{p \times p}$. Again referring to Algorithm 2, $A^T y \in \mathbb{R}^n$ can also be precomputed, where $\lambda_{ADMM} = \tau \max(|A^T y|)$, with $\tau$ a scaling constant. This allows us to break down and roll out the algorithm into discrete steps involving only simple matrix arithmetic.

B. lean PGD

With reference to the lasso depth reconstruction problem in Eq. (1), considering $f(x) = \|y - Ax\|^2_2$ and $g(x) = \lambda_{PGD}\|x\|_1$, the proximal gradient method iterates the value of the reconstructed depth, $x$, as follows,

\[
x^{k+1} := \text{prox}_{\lambda_{PGD}^k}(x^k - \lambda_{PGD}^k \cdot \nabla f(x^k))
\]

where $\lambda_{PGD}^k$ is the step size.

The standard procedure of PGD applied to (7) with proximal projection $\text{prox}_{\lambda_{PGD}^k}(\bullet)$ is described in Algorithm 3.

Algorithm 3. Standard Proximal Gradient Descent [34]

Input: $A, y$

Output: $x$

Initialization: $\text{const} \\{\lambda^0, \beta, e, k_{\max}\}, \text{zeros} \\{u^0\}$
1. for $k = 0$ to $k_{\max}$ do
2. $g_x = A^T Ax - A^T y$
3. while $f(z) > f_k(x^k, y)$ do
4. $z = \text{prox}_{\lambda_k_{PGD}}(x^k, y)$
5. $\lambda_{PGD}^k = \lambda_{PGD}^k \cdot \beta$
6. end while
7. $u^{k+1} = z$
8. if $\|f(x^{k+1}) - f(x^k)\| < e$ then
9. break
10. end if
11. end for

\[
\hat{f}_k(x, y) = f(y) + \nabla f(y)^T \cdot (x - y) + (1/(2 \lambda_{pgd}) \cdot \|x - y\|^2_2).
\]

To improve the efficiency, we again identify the constant components during iteration and pre-compute $A^T A$ and $A^T y$. In Algorithm 3, the gradient step $\lambda_{PGD}^k$ is changed during majorization-minimization, reducing $\lambda_{PGD}^k$ by a fractional factor $\beta$, which introduces uncertainty into the number of iterations. This causes unpredictable timing closure in circuit design. Therefore, we propose a lean PGD algorithm, shown in Algorithm 4, including the essential gradient descent and proximal projection where $W = A^T A$ and $V = A^T y$ are precomputed.

Algorithm 4. lean Proximal Gradient Descent

Input: $W, V$

Output: $x$

Initialization: $\text{const} \\{\lambda^0_{PGD}, \beta^0, \gamma, k_{\max}\}, \text{zeros} \\{u^0\}$
1. for $k = 0$ to $k_{\max}$ do
2. $g_x = W x^k - V$
3. $x^{k+1} = \text{prox}_{\lambda_k_{PGD}}(x^k - \lambda_k_{PGD} \cdot g_x)$
4. $\lambda_{PGD}^k = \lambda_{PGD}^k \cdot \beta^k$
5. $\beta^{k+1} = \min(\gamma \cdot \beta^k, 1)$
6. end for
TABLE 1. Arithmetic complexity.

\[
\begin{array}{lll}
\hline
\text{Operator} & \ell \text{ADMM} & \ell \text{PGD} & d\text{Sparse} \\
\text{ADD/SUB} & 4 \cdot n \cdot (p - 1) + k_{\text{max}} \cdot n \cdot (19 + 4 \cdot (p - 1)) & 4 \cdot n \cdot (p - 1) + k_{\text{max}} \cdot (n \cdot p + 8 \cdot n + 2) & 2p \\
\text{MUL} & 4 \cdot n \cdot p + k_{\text{max}} \cdot n \cdot (5 + 4 \cdot p) & 4 \cdot n \cdot p + k_{\text{max}} \cdot (n \cdot p + n + 1) & 2pn \\
\text{DIV} & n \cdot (4 \cdot p + 5) + 4 \cdot p + 3 & n \cdot (p + 4 \cdot p + 4 \cdot n + 2) + k_{\text{max}} & 4pn \\
\text{MEM} & + k_{\text{max}} \cdot (25 \cdot n + 9 + 4 \cdot n \cdot p + 4 \cdot p) & & \\
\hline
\end{array}
\]

Instead of searching the gradient step $\lambda^{k}_{\text{PGD}}$ dynamically, the majorization-minimization procedure is removed and an approximate method is considered with a predefined gradient step and a fixed updating schedule. By carefully analyzing the $\ell$PGD algorithm, we introduce an additional fractional factor $\gamma$ on $\beta$, which slows down the decrement of $\lambda^{k}_{\text{PGD}}$ and approaches the optimal gradient descent step size gradually. To further improve the timing closure, early termination by checking the objective difference $e$ between iterations is not checked or allowed. By introducing a fixed number of iterations, as with $\ell$ADMM, an efficient implementation can be achieved due to its determinism.

C. PSEUDO-INVERSE LEAST-SQUARES: $d$SPARSE

By distributing the sparse signal recovery across many small blocks of size $n$, the size of the sensing matrix, $A$, is small enough to consider increasing the number of measurements to $p > n$ without a dramatic increase in size or sampling time for a constant block size. This problem is over-determined, but uses a sparse measurement scheme on each laser projection to reduce emission power. In a pattern sequence $A = [A_1, \ldots, A_i, \ldots, A_m]$, every pixel is activated at least once in the sequence to ensure that $A$ is full-column rank [36]. This allows depth recovery using an approximated least squares solution [37]. Let $A_{\text{max}} = (A^T A)^{-1} A^T \in \mathbb{R}^{n \times p}$ with $p > n$, then an image for each signal is simply

\[ x = A_{\text{max}} y. \]  

Assuming a fixed sequence of patterns in $A$, $A_{\text{max}}$ can be pre-computed and stored. We call this approach $d$Sparse [26]. This approach requires more pattern exposure cycles and thus has a slower sampling time than the full compressive formulations, but the sampling time is constant for each approach as defined on a per block basis even if the number of blocks and thus full image resolution increases.

D. DEPTH RECONSTRUCTION

Solving the twin optimization problems in Section II using $\ell$ADMM or $\ell$PGD, the inverse DCT is applied to both the recovered ToF-sum $x_{Qcs}$ and intensity $x_{Ics}$ as shown in Eq. (9).

\[
\begin{cases}
\tilde{x}_{Qcs} = F^{-1}(x_{Qcs}) \\
\tilde{x}_{Ics} = F^{-1}(x_{Ics}).
\end{cases}
\]

By dividing the outcomes of the twin optimization problems, the final depth pixel vector $x_D$ is recovered as shown in Eq. (10).

\[ x_D = \begin{cases}
\frac{x_{Qcs}}{x_{Ics}}, & \text{for } \ell \text{ADMM and } \ell \text{PGD} \\
\frac{x_{Qds}}{x_{Ida}}, & \text{for } d\text{Sparse}.
\end{cases} \]

where $x_{Qds}$ and $x_{Ida}$ represent the ToF-sum and intensity for $d$Sparse.

Investigating the effects of reduced precision, we show how approximate computing (AC) enables a flexible hardware design for a parallel sparse reconstruction unit to enable efficient arrayed 3D imaging sensors for LiDAR applications. Using the small footprint enabled by reduced precision (RP), we next evaluate the balance between resource efficiency and accuracy of reconstruction for all three algorithms when implemented on re-configurable FPGA devices.

IV. AN EVALUATION OF THE PROPOSED APPROACH

A. COMPUTATIONAL COMPLEXITY

To analyse the computational complexity of $\ell$ADMM, $\ell$PGD, and $d$Sparse, the key parameters are the number of measurements, $p$, the signal size, $n$, and the maximum number of iterations, $k_{\text{max}}$. $p$ and $n$ determine the dimensions of the sparse sensing matrix $A$ in Algorithms 2 and 4. The matrix $A_{\text{max}}$ in $d$Sparse is derived from the sensing matrix, where $p > n$ because of oversampling.

The arithmetic operations of addition/subtraction, multiplication, division, and memory addressing are counted separately. By considering the linear algebra computations for the three algorithms, the corresponding complexities of the three algorithms are compared in Table 1.

It is clear that $d$Sparse has the least complexity as a function of $n$, $p$ and $k_{\text{max}}$. $\ell$ADMM and $\ell$PGD are of similar complexity, though $\ell$PGD is slightly less. However, when comparing $d$Sparse to the CS implementations we must take into account that the number of measurements, $p$, is much greater so these calculations do not imply that $d$Sparse will be the most efficient.

Given the same $n$ for depth image reconstruction using all three approaches, the number of parallel blocks is identical. Hence, the computational complexity within a single block is representative of the overall complexity. To further explore the parallel block based reconstruction, the arithmetic complexity for a single block with vector size $n$ is illustrated in Figure 4 where a fixed number of measurements, $p$, and iterations, $k_{\text{max}}$, are considered. For $\ell$ADMM and $\ell$PGD, the number of measurements, $p = 8$, while for $d$Sparse, $p = 48$. For a fair comparison, five iterations have been set for both the $\ell$ADMM and $\ell$PGD algorithms for processing of all scenes. By carefully defining the algorithm parameters, such as $\rho$, $\lambda$,...
\[ \gamma, \text{ and } \beta, \text{ we can approach guaranteed convergence} \ [38], \ [39] \text{for the illustrative block size of } n = 16. \]

It is clear from Table 1 and Figure 4 that the complexity is a linear function of the block vector size \( n \), for fixed \( p \) and \( k_{max} \). For all three cases, this shows the importance of minimising the block size for efficient implementation, provided accurate reconstruction is maintained. The other key observation is that the complexity of the ADMM approach is much higher than the alternative PGD CS approach.

In contrast to solving (1) for \( p = 8 \) or less in parallel for each block (\( n = 16 \)), a full frame, \((n_F = 64 \times 64 = 4096)\) image would require \( p_F > 600 \) measurements, making the reconstruction by PGD or ADMM much more expensive and would require 4 orders of magnitude more storage for \( A \).

Some CS implementations rely on implicit generation of the sensing matrix, \( A \), but add more computational steps [40]. In general, CS techniques will reduce the amount of sampled data but increase the computational complexity in the reconstruction. We try to reduce this impact on complexity, and thus resources, by applying reduced precision, where we assume a fairly small \( 4 \times 4 \) sensing block size with a block image vector size of \( n = 16 \).

Notice that, in Figure 4, although the comparison of complexity in arithmetic and memory operations suggests the PGD is superior given the fixed pre-defined condition of \( A \), the performance and cost are not reflected exactly by the complexity. Detailed analysis is given in following sections.

**B. ACCURACY OF RECONSTRUCTION**

The central argument of this work is that we can achieve almost equivalent quality of depth reconstruction with significant savings in resource consumption using reduced precision. Hence, to justify that argument, we assess first the quality of approximately reconstructed depth images using a number of different scenarios.

We use two synthetic benchmarks, one indoors and at short range of a room with furniture [41], the other outdoors and at longer range simulating an automotive environment [42]. The other histograms that we use are real data collected at our university using a photon counting LiDAR system as part of an earlier study into underwater object detection and classification [43]. We emphasise that the source data in all cases is a 2D matrix of histograms of the form shown in Figure 1.

For the real data, that is what we collect, for the synthetic data we create these from the known depth values using a simulation package that we have used successfully for work on surface detection using deep learning and compressive sensing [5], [44]. The final signal is simulated as a Poisson counting process, i.e. \( h = P(s + 2), \) where \( s \in \mathbb{N}^+ \) contains photon count means with a simulated instrumental similar to the system used in [43] at their respective surface return locations in the histogram [18], [44] and a mean background level of around 2 counts per bin.

We consider all approaches presented in Section III, i.e. the parallel CS /ADMM and /PGD algorithms and the parallel, over-sampled least squares approach, /dSparse, at various precision using floating point (FP), fixed point (FXP) and unum posit (UP) arithmetic. In all cases, full, double precision floating point computation using /dSparse is considered as the baseline comparison for the metrics for quality of reconstruction, i.e PSNR and SSIM [45]. For the CS algorithms, the data are re-sampled using pseudo-random binary patterns. For the over-sampled /dSparse algorithm, random patterns are also used but each pixel is sampled at least once. In all cases, if the precision is reduced below the values shown, the image quality is considered unacceptable.

1) SYNTHETIC DATA

The results for the indoor benchmark scenes [41] are illustrated in Figure 5 for /dADMM \((\rho = 1.2, \alpha = 1.4, \tau = 10^{-9})\) in (c)-(e) , /dPGD \((\gamma = 4.8 \text{ on } \beta = 0.55)\) in (h)-(j) and /dSparse in (m)-(o) respectively.

The first observation is that the results for /dSparse are clearly better than the CS approaches. For example, SSIM is greater than or equal to 0.99 in all cases, compared to 0.75 or less for the CS cases, and the PSNR metric is also much higher, above 60dB compared to a best case of 30.57dB (UP20). Visually, the images are less noisy and show far less “blocking”. The block artefacts in /dADMM and /dPGD are primarily due to the chosen basis and the use of an identical pattern sequence for all blocks. However, the ability to share the pattern sequence across all blocks reduces storage since that pattern can be stored in a single, rather than multiple, different blocks.

As shown, the precision of /dSparse can be reduced to 18 bits for both floating point and unum posit arithmetic and 38 bits for fixed point arithmetic without any noticeable effect. Beyond these points the images degrade noticeably. However, compared to full precision CS depth reconstruction, both /dADMM and /dPGD can reduce the arithmetic bit width to 20 bits for floating point. With fixed point arithmetic, /dPGD can use reduced 22 bit fixed point, 1 bit less than /dADMM. With unum posit /dPGD can be reduced to 22 bits, 2 bits more than /dADMM.

The corresponding results for the outdoor benchmark scenes [42] are illustrated in Figure 6 for /dADMM \((\rho = 1.2, \alpha = 1.4, \tau = 10^{-9})\) in (c)-(e) , /dPGD \((\gamma = 6 \text{ on } \beta = 0.6)\) in (h)-(j) and...
Sparse in (m)-(o) respectively. Broadly, the results are similar to the indoor scenes, as a visual comparison and examination of the SSIM values shows. The PSNR values are again much higher for \( d \) sparse, but we observe that there is a marked difference between the indoor and outdoor simulations. That is because we include a full, optical laser propagation and reflection model in our simulations so that the peak power at longer range is much reduced in comparison with the noise level at the receiver. However, what is important is that the algorithms can reconstruct the depth images at the lower PSNR levels.

**FIGURE 5.** A visual comparison of the final depth map reconstruction using the several data types and precision, using simulated photon count data created from a real depth scene [41]. The image size is \( 64 \times 64 \).

**FIGURE 6.** A visual comparison of the final depth map reconstruction using the several data types and precision, using simulated photon count data created from a real depth scene [42]. The depth image size is \( 128 \times 128 \).
2) REAL LIDAR DATA

These underwater images were all collected in a tank with a stand-off distance of 1.33 metres, slightly less than the total tank length, using a time-correlated single photon counting LiDAR system [17]. Our methodology was identical to the synthetic cases. For the first scene, which consists of a set of circular target blocks on a sandy base, the results are illustrated in Figure 7 for \( \ell \text{ADMM} (\rho = 1.2, \alpha = 1.4, \tau = 10^{-5}) \) in (c)-(e), \( \ell \text{PGD} (\gamma = 4 \text{ on } \beta = 0.6) \) in (h)-(j) and \( \text{dSparse} \) in (m)-(o) respectively. In the second example, which is of a tethered, metal ball bearing (actually simulating an underwater mine), hidden behind foliage, the results are shown in Figure 8 for \( \ell \text{ADMM} (\rho = 1.2, \alpha = 1.4, \tau = 10^{-5}) \) in (c)-(e), \( \ell \text{PGD} (\gamma = 7.5 \text{ on } \beta = 0.45) \) in (h)-(j) and \( \text{dSparse} \) in (m)-(o) respectively.

In general, our observations of the real data confirm the synthetic cases, i.e that the \( \text{dSparse} \) algorithm will produce better results than the CS methods, and that the latter also have more pronounced blocking effects. The differences between the CS methods are slight, as indeed they should be as they use the same objective function to minimise, although the methods are different.

In terms of reduced precision, we again observe that significant reductions in bit width are possible, for example, referring to Figure 7 using \( \text{dSparse} \), an SSIM around 0.99 is retained at 20 bit floating point, 33 bit fixed point and 24 bit unum posit arithmetic. The CS bit reductions are similar, \( \ell \text{PGD} \) retains a SSIM value over 0.40 for 20 bit floating point, 36 bit fixed point, and 26 bit unum posit arithmetic while the SSIM of \( \ell \text{ADMM} \) degrades to around 0.25 for 24 bit floating point, 36 bit fixed point and 30 bit unum posit arithmetic. Similar conclusions are drawn from Figure 8.

In summary, the parallel CS approaches, \( \ell \text{ADMM} \) and \( \ell \text{PGD} \), and the least-squares approach, \( \text{dSparse} \), are all able to reconstruct the depth information for simulated and real photon count data at significantly reduced precision. From the algorithmic point of view, \( \text{dSparse} \) dominates depth reconstruction performance measured by SSIM, as the data is over-sampled and the reconstruction is computed by a direct, least-squares solution to the inverse problem. Yet, both CS approaches retain relatively high reconstructed depth image quality as measured by PSNR when compared to \( \text{dSparse} \). It is also important to re-affirm that a major justification for the use of CS for active LiDAR systems is to reduce the projected power, and thus allow eye-safe operation at longer range than would be possible using a full \( \text{dSparse} \) or other 3D reconstruction method. Given the comparative evaluation of depth reconstruction quality between full and reduced precision, it is clear that the approximate solutions from all three methods are only slightly degraded when compared to corresponding full precision solutions. This enables cost savings in hardware implementations as illustrated in the next section.

V. IMPLEMENTATION ON AN FPGA PLATFORM

Our goals are to save power and memory on a dedicated sensor platform using an FPGA architecture, and to support designs using variable precision. In this section we describe how we design and synthesise the hardware for the three different algorithms, \( \ell \text{ADMM} \), \( \ell \text{PGD} \) and \( \text{dSparse} \). To
support these algorithms, we develop functions using a C++ template for arbitrary approximate algebra operators based on the Xilinx fixed-point [46] and the unum SoftPosit libraries [47]. The customized floating point solutions are based on post-processing of the standard implementation from the Xilinx Vivado High Level Synthesis (HLS) suite. All arithmetic data-types implemented in C++ are synthesized for the Xilinx FPGA UltraScale+ ZCU106 using Vivado 2019.1.

A. APPROXIMATE ALGEBRA MODULES

In Figure 9, the matrix/vector arithmetic modules, including vector addition/multiplication/division/comparison and matrix to vector multiplication, are listed.

The vectors and matrices are buffered through a FIFO data path while the scalar number is transmitted by a single register line data path. The approximate algebraic modules are able to perform scalar arithmetic operations using the different data types and precision. In Figure 9, only the module interfaces are shown, neglecting the external memory addressing of the matrix and array elements.

- **VEC_ADD** performs the addition of two arrays using the same specific data type with pre-defined precision. Since the inputs are two vectors, two FIFO data paths used to stream the data to the scalar arithmetic where the computation is pipelined for each pair of inputs. The output is streamed through a single FIFO data path to successive modules. The same process is operated for **VEC_SUB** and **VEC_DIV**.

- **VEC_SCALAR_ADD** has one array input and one scalar input, both using the same data type with pre-defined precision. Each time an element in the input array is added to the input scalar number, the outcome is streamed through the output FIFO. This process is also applied to **VEC_SCALAR_SUB**, **VEC_SCALAR_MUL**, and **VEC_SCALAR_CMP**.

- **VEC_SCALAR_CMP** does not only compare the elements in the input array but also selects the maximum as the output.

- The **MAT_VEC_MUL** module multiplies a matrix by an array using column-wise element multiplication and accumulation of the same specific data type with pre-defined precision.

All the modules are designed with the Xilinx HLS stream pragmas specifically for the AXIS bus interface. The arithmetic units are shown as generic basic blocks while the different data types (FP, FXP, or UP) and precision are tailored on-demand for different depth reconstruction methods and scenes.
B. APPROXIMATE DEPTH RECONSTRUCTION ARCHITECTURE

First, we consider the CS approaches. By composing the approximate algebra modules of Section V in different combinations, the implemented iterative architectures for Algorithms 2 and 4 are illustrated in Figures 10 and 11 respectively as data flow diagrams. External memory addressing is represented by the arrow links. The input data is read from external memory into the buffer of the corresponding modules while the data is streamed between modules. The output is only written to external memory once the iterations are concluded.

For ADMM and PGD, the CS depth reconstruction architecture is illustrated in Figure 12, where the outcomes from the twin solvers are processed within a parallel data path. Each data path contains a pair of MAT_VEC_MUL units to pre-process the algorithm input and post-process the output by inverse DCT. The outcomes of both data paths are combined in a VEC_DIV unit to generate the final depth values.

C. RESOURCE UTILIZATION AND PERFORMANCE

The resource utilization of the depth reconstruction architectures shown in Figures 12 and 13 are illustrated in Table 2 for a sensing block using the several custom floating point, fixed point and unum posit representations across all scenarios. The comparative metrics are compared to full double precision and a data ratio given by

\[ \text{data ratio} = \frac{p}{n} \cdot \frac{\text{bits}}{64}, \]  

with constant \( p \) and \( n \), where \( \text{bits} \) indicates the total bit width of corresponding arithmetic types.

Using Vivado power estimation tools, the FPGA system has a static power consumption baseline of roughly 0.6 W consistently for all three depth reconstruction approaches. The power savings of dynamic power consumption above this baseline are analyzed.

For ADMM, there are dramatic savings of above 65% in LUT, 90% in DSP, 100% in BRAM using customized floating point (FP) enabling more than 75% power saving and 60% data compressive ratio reduction. The processing latency using customized floating point is reduced by a factor of almost 2 compared to full precision. With fixed point arithmetic (FXP), there are similar resource savings greater than 60% in LUT, 100% in BRAM and around 40% in DSP, enabling around 75% power saving and more than 40% data.
reduction. The processing latency using fixed point is further reduced by a factor of nearly 6. Using unum posit, there is over 70% more LUT usage than full precision while there is 40% in DSP and 100% in BRAM usage reduction. The power consumption is also reduced by over 80% with more than 50% data reduction. It is not as fast as either customized floating point or fixed point, but still gains over 30% processing latency reduction when compared to full precision.

For PGD, there are also significant savings of above 75% in LUT, 85% in DSP, and 100% in BRAM for customized floating point, with over 80% power saving and 65% data compressive ratio reduction. The processing latency is also reduced by around 45%. With fixed point arithmetic, the resource utilization is reduced by over 65% in LUT and 100% in BRAM, however, slightly more DSP usage, around 10%, is introduced. The power consumption is reduced by over 75% while the data ratio is reduced by over 40%. The latency of fixed point arithmetic is reduced by nearly 95% which is over 10x faster than full precision. By using unum posit for PGD, there is slightly 10% more LUT usage and slightly 20% less DSP usage than full precision. The power consumption is reduced by over 85% with above 25% less processing latency.

For dSparse, similar trends are observed. With customized floating point, the resource saving is reduced by 85% in LUT, 80% in both DSP and BRAM, and 87% power saving is achieved with over 67% data ratio and 75% processing latency reductions. By using fixed point arithmetic, the resource saving is reduced by 90% in LUT, 60% in DSP and 73% in BRAM. Power is reduced by over 92%, while the data ratio and processing latency are both reduced by around 50%. The unum posit data type for dSparse saves resource by 45% in LUT, 50% in DSP and 46% in BRAM, which introduces saving of 89% in power and 65% in data ratio. The latency using unum posit is slightly higher than the full precision by over 12%.

Hence, in all cases, significant reductions in resources, data usage, power and latency, are demonstrated. This enables great flexibility in system design for 3D imaging systems with control over resource use and power efficiency. Comparing the three algorithms, the two CS approaches have similar logic and memory resource utilization but dSparse out-performs both in terms of logic resource use and dynamic power. Indeed, dSparse drops to less than 100 mW in the best case. This is due to the much simpler computational operations shown in Figure 13. However, using dSparse, the data addressing is more complex for matrix to vector multiplication and so more BRAM storage units are used at all levels of reduced precision below the full FP64 case. dSparse has a much higher data ratio, which is due to the lengthened data acquisition stage and the increase in the sensing matrix size.

In our FPGA implementations, we have considered processing of blocks of pixels corresponding to the size of the sensing pattern, which is 4 × 4. This allows a maximum frame rate of well above 3 kHz for lADMM, over 7.5 kHz for PGD, and 11 kHz for dSparse if all blocks are processed.
in parallel. Given the image sizes, there are 256 blocks in Figure 5, 1024 blocks in Figure 6, and 2304 blocks in Figures 7 and 8. Depending on the number of parallel processing blocks, the frame rate can be adjusted by trading the overall resources and power consumption. For example, if 32 blocks are processed in parallel for the scenario in Figure 5, this reduces the frame rate to 375 Hz for \( \ell \text{ADMM} \), over 937.5 Hz for \( \ell \text{PGD} \), and 1.375 kHz for \( d\text{Sparse} \). This is comparable to the hypothetical x86 processing times presented in [5] but here we have been able to show synthesised FPGA results that clearly demonstrate the effectiveness of RP in saving resource for the depth reconstruction problem.

E. EFFICIENCY EVALUATION

The efficiency of RP approximated depth reconstruction is also illustrated in Table 2. For the \( 4 \times 4 \) sensing pattern of 16 depth pixels, two major metrics are considered: throughput with reconstructed depth pixels per second and energy consumption with micro Joules per reconstructed depth pixel. By considering the latency time of processing a single sensing block \( (4 \times 4 \text{ pixel block}) \), the system throughput in pixels per second is

\[
\text{Throughput} = \frac{16}{\text{Latency}},
\]

while the energy of pixel processing in \( \mu \text{J} \) per pixel is

\[
\text{Energy} = \left( \text{Latency} \cdot \text{Dynamic Power} \right)/16.
\]

As shown, when working with RP, the efficiency of \( \ell \text{ADMM} \) is not comparable to the other two approaches as it has the lowest throughput and the highest energy consumption for all the different scenes. On the other hand, RP approximated \( \ell \text{PGD} \) using fixed point has the highest throughput, but RP approximated \( d\text{Sparse} \) using fixed point has the advantage of lower overall energy consumption, especially for those synthetic and real LiDAR scenarios in Figures 5(n), 6(n), and 7(n) with only 0.11 \( \mu \text{J} \) per reconstructed pixel. However, the \( \ell \text{PGD} \) algorithm achieves both higher throughput of processing, 1.6 million pixels per second, and lower energy consumption, 0.09 \( \mu \text{J} \) per reconstructed pixel, for the synthetic scenario in Figure 4(i). The highlighted numbers with bold font in Table 2 indicate the top selected design metrics across all three approaches for different scenes.

Notice that all the implementations use the HLS method for the purpose of prototyping comparisons without heavy design optimization which can be potentially improved with fine-grain RTL design. The prototyping outcome indicates that a simpler algorithm does gain advantages in resource utilization and power consumption with approximation, while the throughput and energy efficiency might vary depending on the generated architecture. Nevertheless, one can trade the metrics from HLS outcomes as guidelines to meet the particular design requirements.

VI. CONCLUSION

We have presented the effects of applying reduced precision as an approximate computing technique to depth reconstruction using small scale \( \ell_1 \) (\( \ell \text{ADMM} \)) and \( \ell_1 \) (\( \ell \text{PGD} \)) parallel blocked compressive sensing, and a discrete least-square solver (\( d\text{Sparse} \)).

We have demonstrated significant savings in logic resources and much improved latency using reduced precision floating, and unum posid data types with minimal degradation of reconstruction, in comparison with the commonly used double precision floating point arithmetic. We achieved very fast per block computation times, reducing latency from 0.48 ms to 0.01 ms, and significant power savings, by as much as 95%, in the best cases. In the compressed sensing cases, we achieved additional savings by reducing further the overall data bandwidth. This demonstrates a pathway to dedicated logic for depth computation for resource constrained devices. The approach also shows potential ways of mitigating the computational complexity of compressed sensing techniques, while benefiting from low power, eye-safe sparse illumination and lowering the data bandwidth throughout the computation stack.

REFERENCES

[1] S. Hening, C. Ippolito, K. Krishnakumar, V. Stepanyan, and M. Teodorescu, “3D LiDAR SLAM integration with GPS/INS for UAVs in urban GPS-degraded environments,” in Proc. Int. Conf. AAAI Inf. Syst., 2017, pp. 1–10. [Online]. Available: https://arxiv.org/abs/10.2514/MAIA17
[2] A. Coltekin et al., “Extended reality in spatial sciences: A review of research challenges and future directions,” ISPRS Int. J. Geo-Inf., vol. 9, no. 7, Jul. 2020, Art. no. 439.
[3] F. Heide, S. Diamond, D. B. Lindell, and G. Wetzstein, “Sub-picosecond photon-efficient 3D imaging using single-photon sensors,” Sci. Rep., vol. 8, no. 1, 2018, Art. no. 17726.
[4] J. Tachella et al., “Real-time 3D reconstruction from single-photon LiDAR data using plug-and-play point cloud denoisers,” Nat. Commun., vol. 10, no. 1, Dec. 2019, Art. no. 4984.
[5] A. Allmann, B. Stewart, J. F. Mota, and A. M. Wallace, “Compressive super-pixel LiDAR for high-framerate 3D depth imaging,” in Proc. IEEE Glob. Conf. Signal Inf. Process., 2019, pp. 1–5.
[6] W. Liu, C. Gu, M. O’Neill, G. Qu, P. Montuschi, and F. Lombardi, “Security in approximate computing and approximate computing for security: Challenges and opportunities,” Proc. IEEE, vol. 108, no. 12, pp. 2214–2231, Dec. 2020.
[7] K. Roy and A. Raghunathan, “Approximate computing: An energy-efficient computing technique for error resilient applications,” in Proc. IEEE Comput. Soc. Annu. Symp., 2015, pp. 473–475.
[8] P. Pandey, Q. He, D. Pomplii, and R. Tron, “Light-weight object detection and decision making via approximate computing in resource-constrained mobile robots,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2018, pp. 6776–6781.
[9] A. Ibrahim, M. Osta, M. Alameh, M. Saleh, H. Chible, and M. Valle, “Approximate computing methods for embedded machine learning,” in Proc. 25th IEEE Int. Conf. Electron., Circuits Syst., 2018, pp. 845–848.
[10] A. Agrawal et al., “Approximate computing: Challenges and opportunities,” in Proc. IEEE Int. Conf. Rebooting Comput., 2016, pp. 1–8.
[11] E. A. Webster, L. A. Grant, and R. K. Henderson, “A high-performance single-photon avalanche diode in 130-nm CMOS imaging technology,” IEEE Electron Device Lett., vol. 33, no. 11, pp. 1589–1591, Nov. 2012.
[12] M. U. Nguyen, T. T. Dao, and V. H. Tang, “Efficient depth image reconstruction using accelerated proximal gradient method,” in Proc. 10th Int. Conf. Knowl. Syst. Eng., 2018, pp. 1–6.
[13] A. Rateb, S. Kamiah, and R. A Rashid, “A generic top-level mechanism for accelerating signal recovery in compressed sensing,” Digit. Signal Proces., vol. 69, pp. 237–251, 2017.
[14] A. Wills, A. Mills, and B. Ninnes, “FPGA implementation of an interior-point solution for linear model predictive control,” IFAC Proc. Vol., vol. 44, no. 1, pp. 14527–14532, 2011.
[15] N. M. Gürel et al., “Compressive sensing using iterative hard thresholding with low precision data representation: Theory and applications,” IEEE Trans. Signal Process., vol. 68, pp. 4268–4282, Jul. 2020.
[16] G. A. Howland, D. J. Lam, M. R. Ware, and J. C. Howell, “Photon counting imaging depth mapping,” Opt. Exp., vol. 21, no. 20, Sep. 2013, Art. no. 23822.

[17] P. Chhabra, A. Maccarone, A. McCarthy, G. Buller, and A. Wallace, “Discriminating underwater LiDAR target signatures using sparse multi-spectral depth codes,” in Proc. Sensor Signal Process. Defence, 2016, pp. 1–5.

[18] S. Hernández-Marin, A. M. Wallace, and G. J. Gibson, “Bayesian analysis of LiDAR signals with multiple returns,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 29, no. 12, pp. 2170–2180, Dec. 2007.

[19] A. M. Wallace, J. Ye, N. J. Krichel, A. McCarthy, R. J. Collins, and G. S. Buller, “Full waveform analysis for long-range 3D imaging laser radar,” EURASIP J. Adv. Signal Process., vol. 2010, pp. 1–12, 2010.

[20] A. Halimi, R. Tobin, A. McCarthy, S. McLaughlin, and G. S. Buller, “Restoration of multilayered single-photon 3D LiDAR images,” in Proc. 25th Eur. Signal Process. Conf., 2017, pp. 708–712.

[21] S. M. Fatarwala, I. Gyongy, N. A. W. Dutton, B. R. Rae, and R. K. Henderson, “A reconfigurable 40nm CMOS SPAD array for LiDAR receiver validation,” in Proc. Int. Image Sensor Workshop, 2019, pp. 1–4. [Online]. Available: https://imagesensors.org/2019-papers/

[22] R. K. Henderson et al., “A 256×256 40nm 900m CMOS 3D stacked 120dB dynamic-range reconfigurable time-resolved SPAD imager,” in Proc. IEEE Int. Solid-State Circuits Conf., 2019, pp. 106–108.

[23] D. Donoho, “Compressed sensing,” IEEE Trans. Inf. Theory, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.

[24] E. J. Candès, J. K. Romberg, and T. Tao, “Stable signal recovery from incomplete and inaccurate measurements,” Commun. Pure Appl. Math., vol. 59, no. 8, pp. 1207–1223, Aug. 2006.

[25] E. Candès and J. Romberg, “Sparsity and incoherence in compressed sampling,” Inverse Problems, vol. 23, no. 3, pp. 969–985, Apr. 2007.

[26] A. Allmann, Y. Wu, B. Stewart, and A. M. Wallace, “Accelerated 3D image reconstruction for resource constrained systems,” in Proc. 28th Eur. Signal Process. Conf., 2021, pp. 565–569.

[27] R. Tibshirani, “Regression shrinkage and selection via the lasso,” J. Roy. Statist. Soc., vol. 58, pp. 267–288, 1996.

[28] M. A. T. Figueiredo, R. D. Nowak, and S. J. Wright, “Gradient projection methods for sparse reconstruction,” IEEE J. Sel. Top. Signal Process., vol. 1, no. 4, pp. 586–597, Dec. 2007.

[29] S. Boyd, N. Parikh, E. Chu, and B. Peleato, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” Found. Trends Mach. Learn., vol. 3, no. 1, pp. 1–122, 2011.

[30] G. Flegar et al., “FloatX: A C++ library for customized floating-point arithmetic,” ACM Trans. Math. Softw., vol. 45, no. 4, pp. 1–23, Dec. 2019.

[31] J.-M. Muller et al., Handbook of Floating-Point Arithmetic, 2nd ed. Basel, Switzerland: Birkhäuser, 2018.

[32] S. W. Smith, The Scientist and Engineer’s Guide to Digital Signal Processing. San Diego, CA, USA: California Tech. Publishing, 1997.

[33] J. Gustafson and I. Yonemoto, “Beating floating point at its own game: Posti arithmetic,” Supercomputing Front. Innov., vol. 4, pp. 71–86, 2017.

[34] S. Boyd and L. Vandenberghe, Convex Optimization. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[35] A. Beck and M. Teboulle, “Gradient-based algorithms with applications to signal-recovery problems,” in Convex Optimization in Signal Processing and Communications. Cambridge, U.K.: Cambridge Univ. Press, 2009, pp. 42–88.

[36] M. F. Duarte and Y. C. Eldar, “Structured compressed sensing: From theory to applications,” IEEE Trans. Signal Process., vol. 59, no. 9, pp. 4053–4085, Sep. 2011.

[37] E. J. Candès and J. K. Romberg, “Signal recovery from random projections,” in Computational Imaging III, C. A. Bouman and E. L. Miller, Eds., Int. Soc. Opt. Photon., SPIE, vol. 5674, 2005, pp. 76–88. [Online]. Available: https://doi.org/10.1117/12.600722

[38] B. He and X. Yuan, “On the O(1/n) convergence rate of the douglas–rachford alternating direction method,” SIAM J. Numer. Anal., vol. 50, no. 2, pp. 700–709, 2012.

[39] N. Parikh and S. Boyd, “Proximal algorithms,” Found. Trends® Optim., vol. 1, no. 3, pp. 127–239, 2014.

[40] M. F. Duarte et al., “Single-pixel imaging via compressive sampling,” IEEE Signal Process. Mag., vol. 25, no. 2, pp. 83–91, Mar. 2008.

[41] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, “Indoor segmentation and support inference from RGBD images,” in Proc. Eur. Conf. Comput. Vis., 2012, pp. 746–760.

[42] A. Gaidon, Q. Wang, Y. Cabon, and E. Vign, “Virtual worlds as proxy for multi-object tracking analysis,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 4340–4349.

[43] P. Chhabra, A. Maccarone, A. McCarthy, G. Buller, and A. Wallace, “Discriminating underwater LiDAR target signatures using sparse multi-spectral depth codes,” in Proc. Sensor Signal Process. Defence, 2016, pp. 1–5.

[44] A. Altmann, B. Stewart, and A. M. Wallace, “Deep learning for LiDAR waveforms with multiple returns,” in Proc. 28th Eur. Signal Process. Conf., 2020, pp. 1571–1575.

[45] D. K. I. M. Setati, “PSNR vs SSIM: Imperceptibility quality assessment for image steganography,” Multimedia Tools Appl., vol. 80, pp. 8423–8444, 2020.

[46] Xilinx, “Vivado design suite user guide: High-level synthesis,” Accessed: Oct. 11, 2020. [Online]. Available: https://www.xilinx.com/

[47] S. H. Leong, “NGA SoftPosit,” Accessed: Jul. 03, 2019. [Online]. Available: https://gitlab.com/ceclane/SoftPosit

YUN WU (Member, IEEE) received the BSc degree in electronic and information engineering from Dalian Nationalities University, Dalian, China, in 2003, the MSc degree in circuits and system from Hunan University, Changsha, China, in 2007, the MSc degree in radio frequency communication systems from the University of Southampton, Southampton, U.K., in 2008, and the PhD degree in electronic engineering from Queen’s University Belfast, Belfast, U.K., in 2014. He is currently a research associate with Heriot-Watt University. His research interests include wireless/image signal processing, processor architecture synthesis, and approximate computing.

ANDREAS ALTMANN (Member, IEEE) received the BEng (Hons.) degree in electrical and mechanical engineering from the University of Strathclyde, Glasgow, in 2016 and the Engineering Doctorate (EngD) degree in applied photonics from Heriot-Watt University, Edinburgh, in 2021. Since November 2020, he has been a system architect with STMicroelectronics R&D Ltd., Edinburgh, Scotland. His research interests include efficient signal processing of active imaging systems and computer vision.

BRIAN D. STEWART received the BSc degree in electronics and the PhD degree in computer vision from Dundee University, in 1989 and 1992, respectively. He is currently a design architect with Imaging Division, Edinburgh, Scotland. He was with STMicroelectronics, where he was working on software development, modelling, and image/signal processing.

ANDREW M. WALLACE received the BSc and PhD degrees from the University of Edinburgh, in 1972 and 1975, respectively. He is currently an emeritus professor of signal and image processing with Heriot-Watt University and a chartered engineer with the Institute of Engineering Technology. He has authored or coauthored extensively and has secured funding from the EPSRC, EU, and other sponsors. His research interests include LiDAR and 3D vision, image and signal processing, and accelerated computing. He is a fellow with the Institute of Engineering Technology.