Journal of Signal Processing Systems
https://doi.org/10.1007/s11265-022-01766-3

Efficient Reconfigurable Mixed Precision ℓ_1 Solver for Compressive Depth Reconstruction

Yun Wu[1] · Andrew M. Wallace[1] · João F.C. Mota[1] · Andreas Aßmann[2] · Brian Stewart[2]

Received: 16 December 2021 / Revised: 15 April 2022 / Accepted: 1 May 2022

Abstract
Rapid reconstruction of depth images from sparsely sampled data is important for many applications in machine perception, including robot or vehicle assistance or autonomy. Approximate computing techniques have been widely adopted to reduce resource consumption and increase efficiency in energy and resource constrained systems, especially targeted at FPGA and solid state implementation. Whereas previous work has focused on approximate, but static, representation of data in LiDAR systems, in this paper we show how the flexibility of an arbitrary precision accelerator with fine-grain tuning allows a better trade-off between algorithmic performance and implementation efficiency. A mixed precision framework of ℓ_1 solvers is presented, with compact ADMM and PGD, for the lasso problem, enabling compressive depth reconstruction by varying the precision scaling in single bit granularity during the iterative optimization process. Implementing mixed precision ℓ_1 solvers on an FPGA with a pipelined architecture for depth image reconstruction across various sensing scenarios, over **74%** savings in hardware resources and **60%** in power are achieved with only minor reductions in reconstructed depth image quality when compared to single float precision, while over **10%** saving in hardware resources and power is achieved compared to relative consistently reduced precision solutions.

Keywords Compressive sensing · Depth reconstruction · Mixed precision · Alternating direction method of multipliers · Proximal gradient descent · Field-programmable gate array

1 Introduction

Most autonomous systems perceive the surrounding environment via light detection and ranging (LiDAR) [1]. This active sensing technology computes the distance to objects by measuring, at each pixel, the time-of-flight (ToF) between emitted and reflected photons. Both the emission and detection of photons consume power, which is of considerable concern in sensor and processing unit design. The current requirements on depth and spatial LiDAR resolution, however, make its power consumption prohibitive and significantly limit its application to resource-constrained platforms, including mobile devices with limited battery supply and physical space, drone scene mapping with SLAM [2], augmented reality [3] and mobile robotics [4].

A promising paradigm to reduce power consumption in resource-constrained devices is approximate computing (AC) [5], which has been widely adopted in signal processing [6], robotics [7] and machine learning [8]. One particular AC technique reduces the arithmetic precision of operations by representing numbers with fewer bits, thereby decreasing the computational cost of memory and logical units. As a result, reduced precision (RP), as it is often known, leads to significant savings in energy consumption [9]. In LiDAR applications, a side benefit of saving energy in data processing is a reduction of thermal noise in the photon detection components, which are often near the data processing unit [10].

✉ Yun Wu
y.wu@hw.ac.uk

Andrew M. Wallace
a.m.wallace@hw.ac.uk

João F.C. Mota
j.mota@hw.ac.uk

Andreas Aßmann
andreas.assmann@st.com

Brian Stewart
brian.stewart@st.com

[1] School of Engineering and Physical Sciences, Heriot-Watt University, The Avenue, Scotland EH14 4AS Edinburgh, UK

[2] Imaging Division, STMicroelectronics R&D Ltd., Tanfield, Inverleith Row, Scotland EH3 5DA Edinburgh, UK

Published online: 23 May 2022

The iterative ℓ_1 solver Alternating Direction Method of Multipliers (ADMM) was adopted in [11] for parallel compressive sensing (CS) of Lidar based depth image reconstruction, which allowed high 3D imaging frame rates with reduced laser power, memory use, logic cost, and power consumption. Alternatively, the accelerated proximal gradient descent (PGD) algorithm was adopted in [12] for a similar problem. Several other works have investigated the use of AC for convex optimization. In the context of depth reconstruction, Aßmann et al. [13] and Gürel et al. [14] made resource savings by reduction of precision, whereas Wills et al. [15] and Wu et al. [16] have applied approximate PGD methods to Model Predictive Control (MPC). Very recently, Wu et al. [17] deployed reduced precision on convex optimisation using ADMM and PGD for compressive depth reconstruction, not only to reduce power and resource, but also to obtain a faster implementation. However, in previous work, the approximate precision was fixed and pre-determined through the whole execution cycle.

In contrast, mixed precision computation has the potential to enable even further resource savings with overall low arithmetic error for either dense or sparse iterative computations [18]. Automated precision tuning can improve energy efficiency when employing iterative refinement [19]. However, current mixed and auto-tuned precision algorithms are either built for an instruction-driven architecture, or support only floating-point above single precision. This limits both the energy savings [20], and the application of approximation to smaller scale, resource-constrained platforms.

In this work, a new, mixed precision framework is proposed, supporting both fixed and floating point arithmetic. This is applied to 3D LiDAR CS depth reconstruction using a parallel ℓ_1 solver of the least absolute shrinkage and selection operator (lasso) problem [13]. Specifically, by applying mixed precision scaling to typical ℓ_1 solvers, ADMM and PGD, for compressive depth reconstruction, considerable savings in power, logic and memory resources are achieved.

Our contributions are summarized as follows:

- An ADMM solver using iterative mixed precision implementation.
- A comparative ℓ_1 solver, PGD, with iterative mixed precision.
- A mixed precision framework is created for reconfigurable accelerator generation on an FPGA.
- A comparative study of both ℓ_1 solvers between the existing consistent and the new mixed precision framework is evaluated in terms of depth reconstruction quality, implementation cost, and power consumption.

CS depth reconstruction based on convex optimization and the use of precision scaling are introduced in Sect. 2. In Sect. 3, the proposed framework is illustrated. The results are demonstrated in Sect. 4 with a conclusion in Sect. 5.

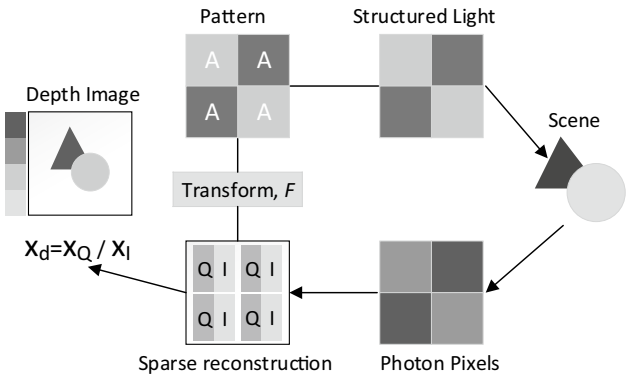


Figure 1 Parallel Depth Reconstruction [13].

2 Background

2.1 Parallel Depth Reconstruction

Using time-correlated single photon counting (TCSPC), LiDAR systems sample depth information with single photon precision at every pixel that can detect photon events [21–24]. The round trip time of a photon, ToF, and thus the distance travelled by each photon, can be accurately measured. To improve the signal-to-noise ratio, it is common to increase the sampling size by accumulating many photon events in a histogram. Recent advancements in solid-state photon count arrays [10, 25, 26] enable high resolution LiDAR imaging. However, as the number of photon count pixels N, i.e., the resolution of the depth image, increases so does the number of raw histogram measurements.

For high resolution LiDAR, it can therefore be challenging to store and process these raw histogram measurements in real-time.

To address this large data volume, [11] has applied compressive sensing [27, 28] to depth imaging and devised a strategy that processes patches of the depth image independently and in parallel, achieving real-time reconstruction. The framework proposed in [11], named checkerboard compressive depth sensing (CBCS), is illustrated in Fig. 1. It uses a random pattern of illumination (structured light) that is reflected from the scene and acquired by photon pixels in a block-based manner. Each block is reconstructed by solving two lasso problems [29]: one for reconstructing a quantity called the *depth-sum*, $x_Q \in \mathbb{R}^{n_B}$, where n_B is the number of pixels in block B, and one for reconstructing a quantity called the *photon count intensity*, $x_I \in \mathbb{R}^{n_B}$. Specifically, given $y_Q \in \mathbb{R}^{m_B}$ (resp. $y_I \in \mathbb{R}^{m_B}$) compressive measurements of the depth-sum (resp. photon count intensity), x_Q and x_I are recovered by solving

$$\underset{x_Q}{\text{minimize}} \ \frac{1}{2}\left\|y_Q - \overline{A}x_Q\right\|_2^2 + \lambda\left\|Fx_Q\right\|_1 \tag{1}$$

$$\underset{x_I}{\text{minimize}} \ \frac{1}{2}\left\|y_I - \overline{A}x_I\right\|_2^2 + \lambda\left\|Fx_I\right\|_1, \tag{2}$$

where $\overline{A} \in \{0,1\}^{m_B \times n_B}$ is a known binary matrix that encodes the active pixels in each block for each measurement in y_Q or y_I, and $F \in \mathbb{R}^{n_B \times n_B}$ is a sparsifying matrix, e.g. a DCT matrix, that we assume is invertible. $\lambda > 0$ is a regularisation parameter, and $\|\cdot\|_2$ and $\|\cdot\|_1$ are, respectively, the ℓ_2- and ℓ_1-norms. After recovering these quantities, the final depth image at block B is computed by dividing x_Q by x_I pointwise: $x_D = x_{Q\cdot}/x_I \in \mathbb{R}^{n_B}$. The work in [11] also proposed a post-processing strategy to remove blocking effects. The problems in (1) are ubiquitous in signal processing and can be solved efficiently via ADMM [30] and PGD [31].

2.2 Mixed Precision and Tuning

Mixed precision combines the accelerated and less resource intensive processing of lower precision with the greater accuracy of higher precision arithmetic. It is normally deployed for iterative refinement using floating point data formats [32, 33], for example using three levels of precision [34]. Customized precision tuning instruments have been employed based on specific input sets [19]. Such precision scaling has gained increasing interest for computation-intensive applications, such as deep learning, saving computation and data storage [35] using both *floating point*

$$-1^{\mathrm{S}} \times \mathrm{M} \times 2^{\mathrm{E}\text{-}127}, \tag{3}$$

where S is the sign, M the mantissa, and E the exponent; and *fixed point*

$$-1^{\mathrm{S}} \times (\mathrm{I} + \mathrm{F}), \tag{4}$$

where S is the sign, I the integer value as sum of 2^b, and F the fraction value as the sum of 2^{-b} from each bit position b depending on the 2's complement format [36].

Different arithmetic types yield different dynamic ranges. Floating point with nonlinear binary representation uses less bits, while fixed point enables simpler binary operations but requires more bits. Hence, the various precisions in both floating and fixed point affect not only the algorithm's performance, but also the embedded implementation.

By adjusting the combination of mixed precision arithmetic at compile-time, static precision tuning is performed once during the system design phase [34]. Such a tuning process can be considered as part of wider hardware/software co-design approaches, profiling the application data and code to compare with empirical measurement before running the system. On the other hand, dynamic precision tuning is invoked multiple times during the execution of the algorithm at run-time, which can incur some overhead [19].

We develop a framework for CS depth reconstruction, using ADMM and PGD to solve lasso. Its key aspect is the mixed precision design of both the arithmetic data type and the binary bit width, appropriate to the performance requirements of the sensing environment. This provides an alternative to our earlier work [17], which used constant precision, and results in a significantly more energy efficient implementation.

3 Mixed Precision Framework

Optimization-based algorithms for lasso are iterative, with the same sequence of steps occurring at each iteration. Traditionally, all computations are performed with the same precision. Here, instead, we consider the case in which different iterations use different precision. We focus firstly on ADMM, secondly on PGD. Then, we present the design flow of our mixed precision accelerator.

3.1 Mixed Precision ADMM

We consider the following reformulation of the problem in (1 and 2):

$$\begin{aligned} &\underset{x,z}{\text{minimize}} \ \tfrac{1}{2}\left\|y - Ax\right\|_2^2 + \lambda_{\text{ADMM}}\|z\|_1 \\ &\text{subject to } x - z = 0, \end{aligned} \tag{5}$$

where we define $A := \overline{A}F^{-1} \in \mathbb{R}^{m \times n}$. Each iteration of ADMM [30] applied to (5) consists of

$$x^{k+1} = \left(A^\top A + \rho I_n\right)^{-1}\left[A^\top y + \rho(z^k - u^k)\right] \tag{6}$$

$$z^{k+1} = S_{\lambda_{\text{ADMM}}/\rho}\left(u^k + x^{k+1}\right) \tag{7}$$

$$u^{k+1} = u^k + x^{k+1} - z^{k+1}, \tag{8}$$

where $\rho > 0$ is the augmented Lagrangian parameter, $I_n \in \mathbb{R}^{n \times n}$ the identity matrix, $u \in \mathbb{R}^n$ a dual variable, and $S_\lambda(\cdot)$ the soft-thresholding operator applied componentwise: for $v \in \mathbb{R}$, $S_\lambda(v) = v - \lambda$ when $v \geq \lambda$, $S_\lambda(v) = v + \lambda$ when $v < -\lambda$, and $S_\lambda(v) = 0$ otherwise. If the parameter ρ is constant throughout the iterations, the matrix $A^\top A + \rho I_n$ and its inverse can be precomputed. This can be done efficiently via the matrix inverse lemma [37],

$$\left(A^\top A + \rho I_n\right)^{-1} = \frac{1}{\rho} I_n - \frac{1}{\rho^2} A^\top \left(I_m + \frac{1}{\rho} A A^\top\right)^{-1} A \tag{9}$$

and by computing the inverse of the $m \times m$ matrix in (9) via its Cholesky decomposition, $I_m + (1/\rho)AA^\top = LL^\top$, where $L \in \mathbb{R}^{m\times m}$ is lower triangular. The quantity $g := A^\top y$ can also be pre-computed. In our context, such pre-computations have the additional benefit of avoiding unnecessary accuracy loss when using mixed precision during the iterations. The resulting algorithm (with fixed precision) is shown in Algorithm 1, from which we can see that each iteration requires $O(n + m^2)$ arithmetic operations. Step 4 of Algorithm 1 differs from (7) in that it uses over-relaxation, parameterized by $0 < \alpha < 2$, which can improve convergence [30].

Algorithm 1 ADMM for lasso in Eq.(5)

Input: A, L, $\kappa = \lambda_{\text{ADMM}}/\rho$, $g = A^\top y$, $k_{\max}$, α
Initialization: $z^0 = u^0 = 0_n$
1: **for** $k = 0, 1, \ldots, k_{\max}$ **do**
2: $\quad q^{k+1} = g + \rho(z^k - u^k)$
3: $\quad x^{k+1} = \frac{1}{\rho} q^{k+1} - \frac{1}{\rho^2} A^\top (L^\top)^{-1} L^{-1} A q^k$
4: $\quad z^{k+1} = S_\kappa(u^k + \alpha x^{k+1} + (1-\alpha) z^k)$
5: $\quad u^{k+1} = u^k + x^{k+1} - z^{k+1}$
6: **end for**
7: **return** $x^{k_{\max}+1}$

Algorithm 2 Mixed precision ADMM for lasso

Input: $g = A^\top y$, $H = A^\top L^{\top^{-1}} L^{-1} A$, $k_{\max}$
Initialization: $z^0 = u^0 = 0_n$, $l_0 = 0$
1: **for** $k = 0, 1, \ldots, k_{\max}$ **do**
2: $\quad (g_c, H_c, z_c^k, u_c^k) = \text{cast}_c^{(l_k)}(g, H, z^k, u^k)$
3: $\quad (z^{k+1}, u^{k+1}) = *\text{mixiter}^{(l_k)}(g_c, H_c, z_c^k, u_c^k)$
4: $\quad l_k \leftarrow l_k + 1$
5: **end for**
6: **return** $z^{k_{\max}+1}$

Algorithm 2 implements Algorithm 1 using mixed precision. It takes as input the constant vector $g = A^\top y$ and matrix $H = A^\top L^{\top^{-1}} L^{-1} A$, and the maximum number of iterations $k_{\max}$. At each iteration k, the algorithm recasts the constants g and H and the variables z^k and u^k to the precision required at the current iteration. In software such a function is implemented conceptually as a data type cast, while in hardware it is implemented through data path trimming, where the extra data path are unconnected between mismatched bit-width.

Algorithm 3 $(\hat{z}, \hat{u}) = *\text{mixiter}(g, H, z, u)$

Input: g, H, z, u
Initialization: constants $\alpha, \hat{\alpha}, \rho, \hat{\rho}, \kappa$
1: $q = g + \rho \cdot (z - u)$
2: $x = \alpha \cdot (\hat{\rho} \cdot q - H \cdot q) + \hat{\alpha} \cdot z$
3: $z = \max\{0, (x + u) - \kappa\} - \max\{0, -(x + u) - \kappa\}$
4: $u = u + x - z$
5: **return** z, u

These variables are then used by the mixed precision function $*$ mixiter, described in Algorithm 3, which performs the same steps as each iteration of Algorithm 1, but with precision specified by l_k. This variable represents the index to a set of static functions, each of which is written for a predefined precision, e.g., floating point with 32 bits or fixed point with 30 bits. Each of these functions uses the precomputed constants $\alpha, \hat{\alpha} := 1 - \alpha$, $\rho, \hat{\rho} := 1/\rho$, and $\kappa := \lambda_{\text{ADMM}}/\rho$. The initial index l_0 is found based on the maximum ϵ representing the precision loss given a certain threshold. This process is described in Sect. 3.2. During the iterations, l_k is modified to meet the demanding finer accuracy of the algorithm, increasing its precision and changing the elementary bit width gradually. We fixed the number of iterations to $k_{\max} = 5$, determined empirically, as no further improvement on reconstruction of depth has been observed beyond this point [17]. In the particular case of parallel computation using small blocks, 4×4 pixels, ADMM converges in very few iterations.

3.2 Mixed Precision PGD

We now consider the application of proximal gradient descent (PGD) to (1). Each of those problems can be written as

$$\underset{x}{\text{minimize}} \;\; \frac{1}{2}\|y - Ax\|_2^2 + \lambda_{\text{PGD}}\|x\|_1 , \tag{10}$$

where $A := \overline{A}F^{-1} \in \mathbb{R}^{m\times n}$. Defining the convex differentiable function $g(x) := (1/2)\|y - Ax\|_2^2$, the convex function $h(x) = \lambda_{\text{PGD}}\|x\|_1$, and their sum $f(x) = g(x) + h(x)$, PGD applied to (10) yields the iterative shrinkage thresholding (ISTA) algorithm [31, 38]:

$$\begin{aligned} x^{k+1} &= \text{prox}_{\alpha_k h}(x^k - \alpha_k \nabla g(x^k)) \\ &= S_{\alpha_k \lambda_{\text{PGD}}}\left(x^k - \alpha_k A^\top A x^k + \alpha_k A^\top y\right), \end{aligned} \tag{11}$$

where $\alpha_k \geq 0$ is the step size at iteration k, and $\text{prox}_{\alpha_k h}(\cdot)$ the proximal operator of $\alpha_k h$ which, in this case, is the soft-thresholding operator $S_{\alpha_k \lambda_{\text{PGD}}}$, defined in Sect. 3.1. PGD converges whenever $0 < \alpha_k \leq 1/L$, where L is the Lipschitz constant of the gradient of g, i.e., $\nabla g(x) = A^\top A$. That is, L can be found as the maximum eigenvalue of $A^T A$. We use a constant stepsize: $\alpha_k = \alpha = 1/L$.

Algorithm 4 PGD for lasso in Eq.(10)

Input: A, $g = A^\top y$, λ_{PGD}, $k_{\max}$
Initialization: $x^0 = 0_n$, $\alpha = 1/\max \text{eig}\,(A^\top A)$, $g_\alpha = \alpha g$, $W = \alpha A^\top A$, $\kappa = \alpha \lambda_{\text{PGD}}$

1: **for** $k = 0, 1, \ldots, k_{\max}$ **do**
2: $\quad v^{k+1} = W x^k - g_\alpha$
3: $\quad x^{k+1} = S_\kappa\left(x^k - v^{k+1}\right)$
4: **end for**
5: **return** $x^{k_{\max}+1}$

The general algorithm is described in Algorithm 4. It takes as input the matrix A, the vector $g := A^\top y$, the regularizer constant λ_{PGD}, and the maximum number of iterations $k_{\max}$. After computing the stepsize α, it precomputes the vector $g_\alpha := \alpha g$, the matrix $W := \alpha A^\top A$, and the threshold $\kappa := \alpha\lambda_{\text{PGD}}$. Steps 2 and 3 then implement exactly (11).

Algorithm 5 Mixed precision PGD for lasso

Input: A, $g = A^\top y$, λ_{PGD}, $k_{\max}$
Initialization: $x^0 = 0_n$, $\alpha = 1/\max \text{eig}\,(A^\top A)$, $g_\alpha = \alpha g$, $W = \alpha A^\top A$, $\kappa = \alpha\lambda_{\text{PGD}}$, $l_0 = 0$

1: **for** $k = 0, 1, \ldots, k_{max}$ **do**
2: $\quad (W_c, (g_\alpha)_c, x_c^k) = \text{cast}_c^{(l_k)}(W, g_\alpha, x^k)$
3: $\quad x^{k+1} = *\text{mixiter}^{(l_k)}(W_c, (g_\alpha)_c, x_c^k)$
4: $\quad l_k = l_k + 1$
5: **end for**
6: **return** $x^{k_{\max}+1}$

Algorithm 6 $\widehat{x} = *\text{mixiter}(W, g, x)$

Input: W, g, x
Initialization: constant κ
1: $v = Wx - g$
2: $x = \max\{0, (x - v) - \kappa\} - \max\{0, -(x - v) - \kappa\}$
3: **return** x

Algorithm 5 implements Algorithm 4 using mixed precision. Both algorithms have the same inputs and initialization, except that Algorithm 5 now also initializes the function pointer with index l_k, which specifies a static function $*\,\text{mixiter}^{l_k}(\cdot)$ that performs the main computations of PGD with a given precision. The order of these functions, which are accessed via l_k, determines the sequence of precisions, maintained through combined search of various bit widths based on the precision loss of the precomputed parameters and a post-search fine tuning. As in mixed precision ADMM, all the relevant variables in Algorithm 5 are cast to the required precision and then used in the $*$ mixiter function indexed by l_k. cast_c only demonstrates the arithmetic precision transformation at the software level, and is not an overhead in hardware design at the appropriate precision. In this work, we only consider mixed precision within the same arithmetic type. As in mixed precision ADMM, we set the maximum number of iterations of PGD to $k_{\max} = 5$.

3.3 Mixed Precision Accelerator

We developed the semi-automated framework illustrated in Fig. 2 based on the Matlab and Xilinx Vivado toolsets. It quickly prototypes an approximate accelerator with mixed-precision arithmetic on reconfigurable platforms. To support mixed precision design, we created an approximate generic linear algebra library calling third-party arithmetic types, which are based firstly on a customized floating-point library FloatX [32] and secondly on the Xilinx fixed-point library [39]. The library was established with C++ templates in the header only, which allows later specific arithmetic type allocation. Its major features are:

- General (non-symmetric) real value operations.
- Arbitrary bit width floating- and fixed-point arithmetic.
- Basic vector/matrix algebra arithmetic (addition, subtraction, multiplication, division, inversion).
- Triangular factorization (LU, and Cholesky decomposition).
- Orthogonal factorization (QR decomposition).

Based on the approximate linear algebra library, a user-defined kernel, which indicates the specific portion of signal processing application to be the target for approximation, is considered as input in Fig. 2. By compiling the input source, a list of various iterative functions are created using different arithmetic precisions, as expressed in both Algorithms 3 and 6. The source compiler is developed based on the Matlab MEX compiler API where the Design Space Exploration (DSE) with precision adaptation is evaluated based on the input pre-computed data.

The quantization effect of the data from single-precision floating point (32 bits, called full precision in the later context of this paper) to reduced precision is considered as the criterion for precision scaling. Specifically, the normalized absolute difference is used for floating point (FP) arithmetic while the absolute difference is adopted for fixed point

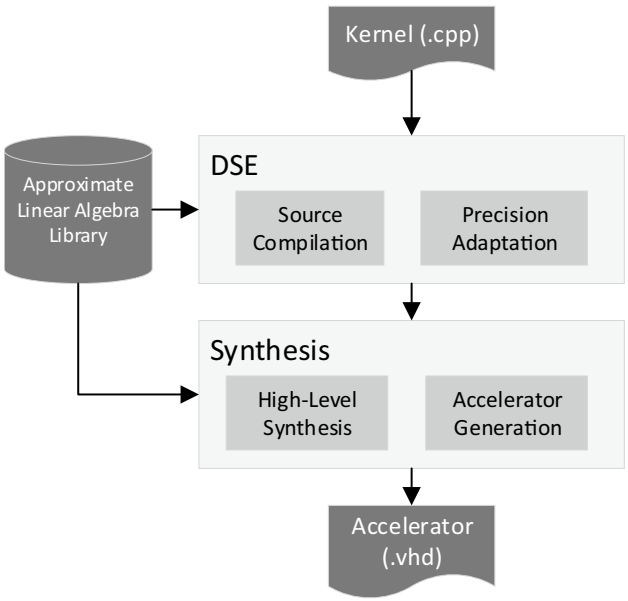


Figure 2 Mixed Precision Design Flow.

(FXP) arithmetic due to the different functional relationships between the binary representations and decimal values. The most representative values, the maximum and minimum of those pre-computed parameters, are picked for the numerical analysis.

To measure the effects of different FP quantizations, we use the normalized absolute difference

$$\epsilon = \frac{\left\| v - \text{cast}_c(v) \right\|_1}{\|v\|_1}, \tag{12}$$

where $v \in \mathbb{R}^n$ is a vector and $\|\cdot\|_1$ the ℓ_1-norm. To measure the effects of different FXP quantizations, we simply use the absolute difference

$$\epsilon = \left\| v - \text{cast}_c(v) \right\|_1. \tag{13}$$

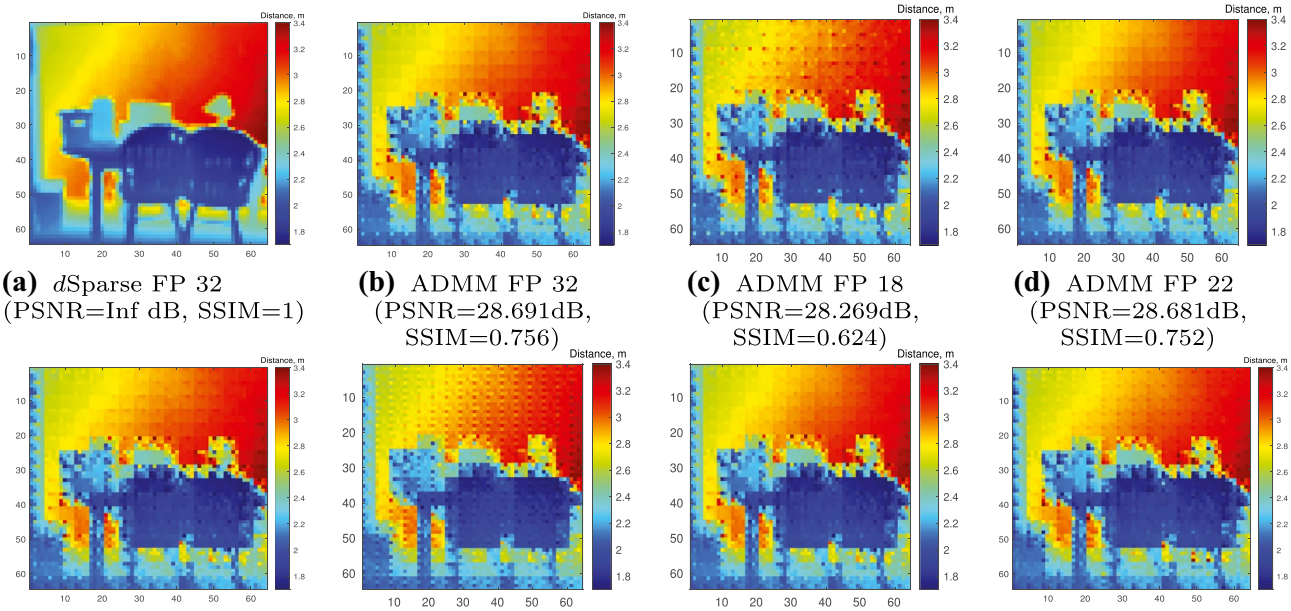
We guarantee that in either (12) or (13), we always have $\epsilon \leq \omega = 5 \times 10^{-3}$. Different arithmetic types will yield different such ϵ. Using exhaustive search, we select the bit width of different parts in the binary arithmetic format for either FP or FXP as the smallest that satisfies $\epsilon \leq \omega$. The representation with that bit width is then indexed by l_0 in Algorithms 2 and 5.

The kernel with chosen precision is adopted for both the integer and fraction part representations to achieve a relatively close reconstruction performance to full precision, where heuristic fine tuning is performed at the DSE stage. The accelerator is generated with a Hardware Description Language (HDL) by calling Xilinx High-Level Synthesis (HLS) tools within the Matlab Framework. With combined algorithm evaluation and hardware implementation in a loop, it can quickly prototype mixed precision design on reconfigurable platforms.

4 Evaluation

Both ℓ_1 solvers, ADMM and PGD, using either floating (FP) or fixed-point (FXP) arithmetic, are illustrated and evaluated for compressive depth reconstruction on two synthetic and two real underwater sensing datasets. Reconstruction of the depth image using *d*Sparse [13] (a non-compressive approach with oversampling) with full, single floating point (FP 32 bits) precision is adopted as a benchmark for comparison.

The FP 32 ADMM or PGD solutions are considered as the baseline, while the identified highest accuracy in the pre-scheduled list is considered as the most suitable reduced precision solution. Both are adopted for comparisons of recovered image fidelity as well as implemented hardware clock speed, resource usage and power consumed.



(a) *d*Sparse FP 32 (PSNR=Inf dB, SSIM=1)
(b) ADMM FP 32 (PSNR=28.691dB, SSIM=0.756)
(c) ADMM FP 18 (PSNR=28.269dB, SSIM=0.624)
(d) ADMM FP 22 (PSNR=28.681dB, SSIM=0.752)
(e) ADMM FP 18→22 (PSNR=28.669dB, SSIM=0.747)
(f) ADMM FXP 20 (PSNR=28.128dB, SSIM=0.601)
(g) ADMM FXP 24 (PSNR=28.735dB, SSIM=0.758)
(h) ADMM FXP 20→24 (PSNR=28.501dB, SSIM=0.751)

Figure 3 Short range dataset [40]: recovered depth images using **a** *d*Sparse, consistent FP 32 bit); and ADMM solver with **b** consistent FP 32 bit; **c** consistent FP 18 bit; **d** consistent FP 22 bit; **e** mixed precision from FP 18 to 22 bit for each iteration; **f** consistent FXP 20 bit; **g** consistent FXP 24 bit; **h** mixed precision from FXP 20 to 24 bit for each iteration.

4.1 Depth Reconstruction Accuracy

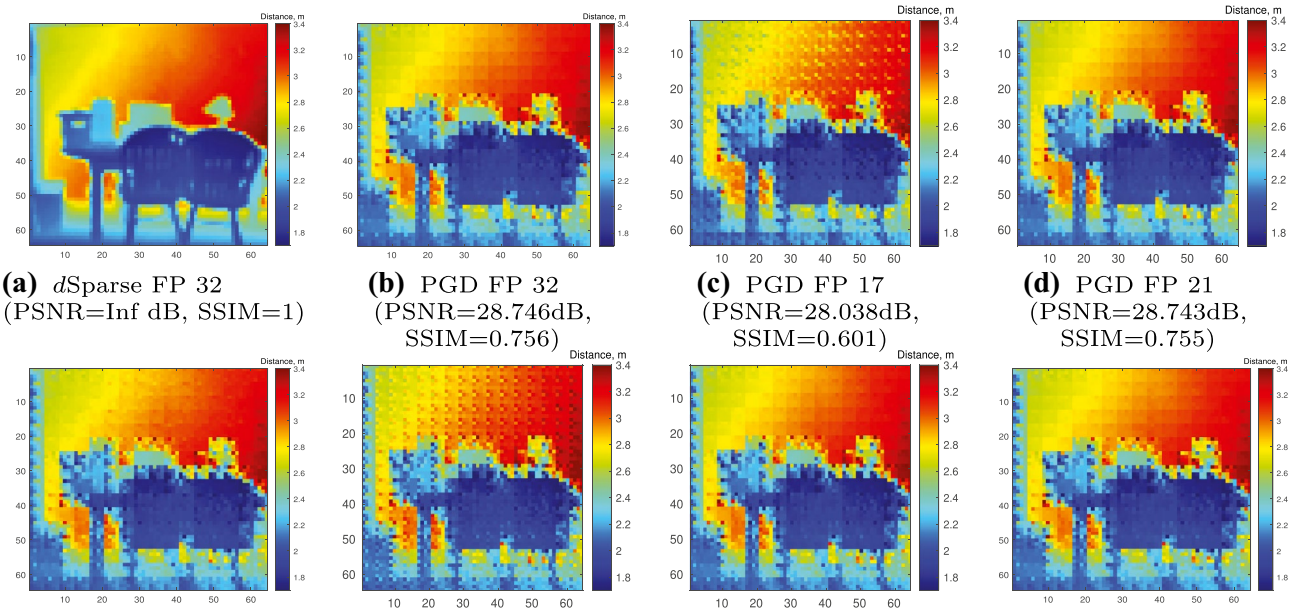
The Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) [41] are both considered as quality metrics, showing the fidelity of reconstruction and the perceived quality based on the pixel-wise absolute difference for normalised similarity scaling. Four different sensing environments are adopted to evaluate the quality of depth reconstruction with mixed precision: short range [40] and long range [42] synthetic benchmarks, and real LiDAR data for two very short range, underwater scenes [43]. In the short range scenario, the detailed mixed precision phenomenon is evaluated firstly against the reconstructed depth image quality for both PSNR and SSIM. Then mixed precision is applied to the other three scenarios, where the corresponding depth reconstruction accuracy is also evaluated.

- *Scene 1: short range*

  By using the design flow for different scenes at different stand-off distances and dynamic ranges, mixed FP from 18 to 22 bits, and FXP from 20 to 24 bits are employed for the short range synthetic scene with distances ranging from 1.8 to 3.4 meters. Figures 3 and 4 illustrate the depth reconstruction accuracy against various consistent and mixed precision for both ℓ_1 solvers, ADMM and PGD. The ground truth is set as *d*Sparse using single precision floating point with 32 bits, where its PSNR and SSIM are infinity and 1 since the comparison is to itself.

  The recovered depth images using the ADMM solver with FP arithmetic are shown in Fig. 3b–e. The PSNR and SSIM are 28 dB and 0.75 respectively for the baseline FP 32 bits (Fig. 3b), while, comparatively, the worst (FP 18) and best (FP22) quality of the pre-scheduled consistent FP precision are illustrated in Fig. 3c, with 0.15 loss in SSIM, and Fig. 3d, with similar PSNR and SSIM. Increasing the bit-width is shown by →, the mixed FP precision (FP 18→22 bits) in Fig. 3e shows only 0.022 dB loss in PSNR and 0.009 loss in SSIM. It has even slight better PSNR than the consistent FP 22 solution. Corresponding depth images recovered by using the FXP ADMM solver are shown in Fig. 3f and g. Similarly, the worst case (FXP 20) in Fig. 3f is degraded in both PSNR and SSIM while the best case (FXP 24) in Fig. 3h has even better PSNR and SSIM than the baseline in Fig. 3b. Using mixed precision FXP 20→24 bits, there is a PSNR loss of 0.19 dB and a 0.004 loss in SSIM.

  Similarly, Fig. 4b–e illustrate the recovered depth image using the PGD solver with FP, while Fig. 4f–h are those using FXP. The baseline in Fig. 4b achieves the same SSIM as the FP baseline with slightly better PSNR. The worst case (FP 17) of pre-scheduled, consistent FP in Fig. 4c degrades in both PSNR and SSIM, while the



(a) *d*Sparse FP 32 (PSNR=Inf dB, SSIM=1)
(b) PGD FP 32 (PSNR=28.746dB, SSIM=0.756)
(c) PGD FP 17 (PSNR=28.038dB, SSIM=0.601)
(d) PGD FP 21 (PSNR=28.743dB, SSIM=0.755)
(e) PGD FP 17→21 (PSNR=28.729dB, SSIM=0.749)
(f) PGD FXP 18 (PSNR=27.791dB, SSIM=0.576)
(g) PGD FXP 22 (PSNR=28.235dB, SSIM=0.737)
(h) PGD FXP 18→22 (PSNR=28.237dB, SSIM=0.739)

Figure 4 Short range dataset [40]: recovered depth images using **a** *d*Sparse with single precision, FP 32 bit; and PGD solver with **b** consistent FP 32 bit; **c** consistent FP 17 bit; **d** consistent FP 21 bit; **e** mixed precision from FP 17 to 21 bit for each iteration; **f** consistent FXP 18 bit; **g** consistent FXP 22 bit; **h** using mixed precision from FXP 18 to 22 bit for each iteration.

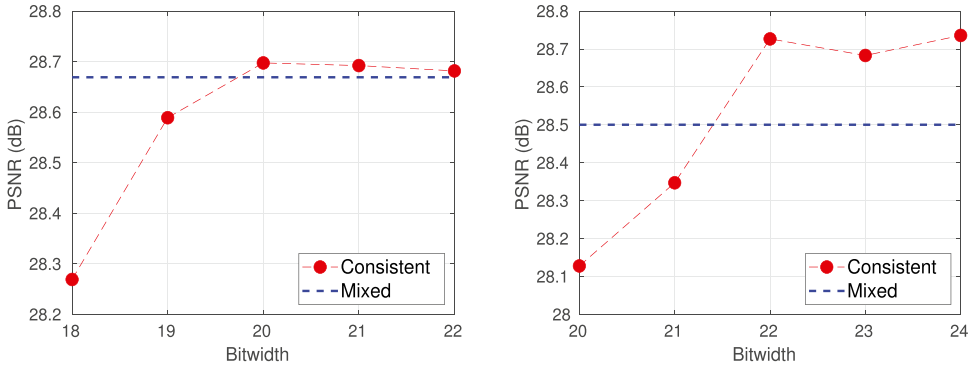


Figure 5 ADMM:PSNR v.s. Bitwidth (Short Range Dataset).

best case in Fig. 4d is almost the same as the baseline. The worst case (FXP 18) of pre-scheduled, consistent FXP has even better PSNR than the baseline but is much worse in SSIM. Both mixed FP and FXP precision show minor loss in both PSNR and SSIM.

To further illustrate the use of mixed precision, PSNR and SSIM are plotted against bit width for both ADMM and PGD using FP and FXP representations in Figs. 5–8. The consistent precision, shown as a red line, refers to the use of the same precision over all iterative computations, and is compared to the use of mixed precision shown as blue line.

As shown in Figs. 5 and 6, for the ADMM solver, the PSNR and SSIM values using mixed precision match the highest, consistent precision in either FP or FXP. An exception is the PSNR of mixed FXP precision in Fig. 5b, which is gradually degraded as the bit width decreases.

Similar trends exist for the PGD solver as shown in Figs. 7 and 8, where all the PSNR and SSIM values for mixed FXP precision match those of the highest consistent precision in either FP or FXP, degrading with decreasing bit width.

Summarising, from Figs. 5–8 we observe that the mixed precision does achieve similar fidelity of the reconstructed depth image when compared to the highest pre-scheduled precision, and to the baseline single floating point precision. However, there are differences between the FP and FXP representations during the transition from low to high precision in the pre-scheduled mixed precision list. The FP representation has a smooth transition for both PSNR and SSIM. Due to the computational uncertainty of truncating errors, the transition of PSNR is slightly bumpy for the FXP representation, but the transition of SSIM is a smooth fucntion of the FP representation.

- *Scene 2: long range*

  Using both ADMM and PGDs, mixed FP from 16 to 20 bits, and FXP from 30 to 34 bits are adopted for the long range, synthetic scene with distances ranging from 0 to 100 meters. Figure 9 illustrates the comparisons of depth reconstruction. The baselines of the reconstructed depth image using ADMM and PGD solvers are illustrated in Fig. 9b and e. The PSNR using the ADMM solver is slightly better than the PGD solver by 0.1 dB, while the SSIM using the ADMM solver is slightly worse by 0.07.

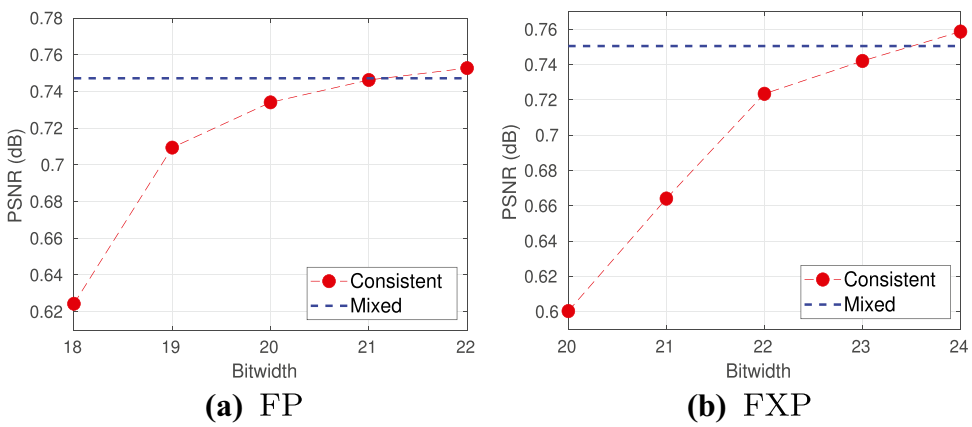


Figure 6 ADMM: SSIM v.s. Bitwidth (Short Range Dataset).

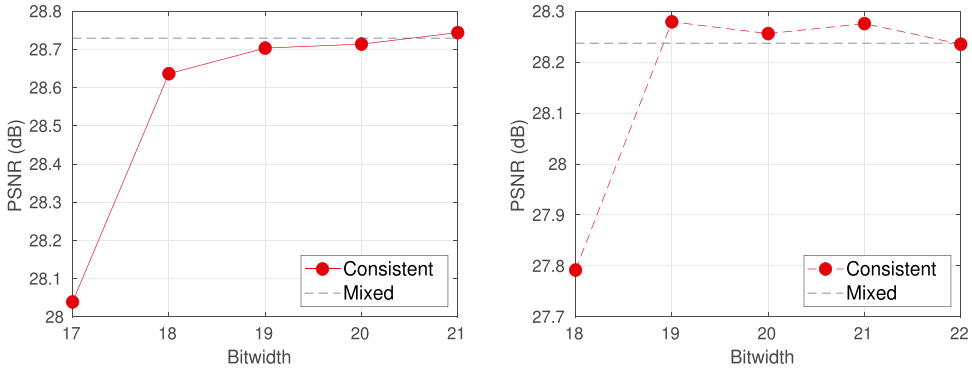


Figure 7 PGD: PSNR v.s. Bitwidth (Short Range Dataset).

With mixed precision using the ADMM solver, there is a 0.2 dB loss of PSNR for FP from 18 to 22 bits and 0.021 dB loss of PSNR for FXP from 28 to 32 bits. Using the PGD solver, the mixed FP precision from 18 to 22 bits shows a slightly better PSNR than the baseline, and than mixed FXP precision from 30 to 34 bits. The SSIM with both FP and FXP mixed precision are both slightly worse than the baseline by about 0.006.

Summarising, evaluation of the use of mixed precision in the long range, as in the short range synthetic scenario, confirms the comparable fidelity of reconstructed depth image to both the baseline and the consistent precision solutions. Further evaluation on real data is now considered.

- *Scene 3: underwater A*

  This scene has distances ranging from 27 to 39 centimeters. Mixed FP from 18 to 22 bits using both the ADMM and PGD solvers in Fig. 10c and f, FXP from 28 to 32 bits using the ADMM solver in Fig. 10d and FXP from 30 to 34 bits using the PGD solvers in Fig. 10g are adopted.

  There is a slight PSNR degradation of about 0.2 dB using ADMM with mixed FP precision, and an almost identical PSNR with mixed FXP precision. SSIM is degraded by around 0.01 for both mixed FP and FXP precision using ADMM. For mixed precision using the PGD solver, the PSNR with both mixed FP and FXP precision is about 0.1 dB better than the baseline, while the SSIM is slightly degraded by 0.005.

- *Scene 4: underwater B*

  This scene has distances ranging from 31.4 to 31.8 centimeters. Mixed FP from 23 to 27 bits using ADMM in Fig. 11c, FP from 19 to 23 bits using PGD in Fig. 11f, FXP from 32 to 36 bits using the ADMM solver in Fig. 11d and FXP from 30 to 34 bits using the PGD solver are illustrated in Fig. 11g. For FP and FXP mixed precision using both the ADMM and PGD solvers, the PSNR amd SSIM are always maintained at a similar level to the baseline FP 32 bits.

  Hence, from our experiments on diverse synthetic and real data, illustrated in Figs. 3–11, we can make the following observations.

i The use of reduced precision in both FP and FXP representations leads to depth reconstruction that has considerable fidelity to the full precision solutions.

ii The use of mixed precision, introduced in this paper, also achieves minor losses in PSNR and SSIM when

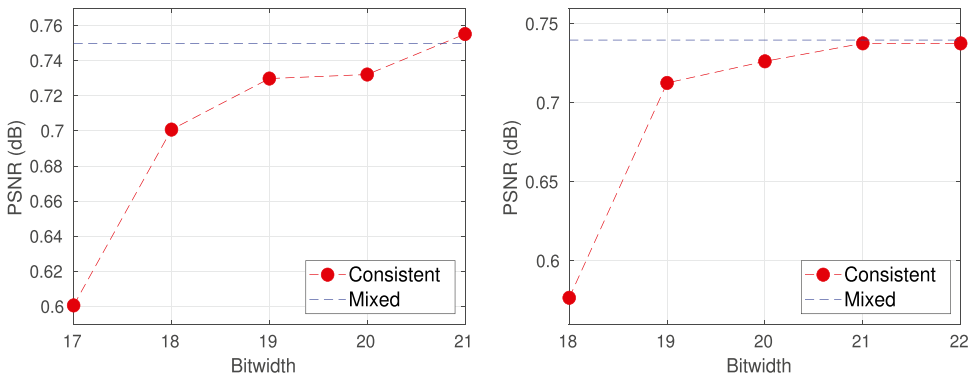
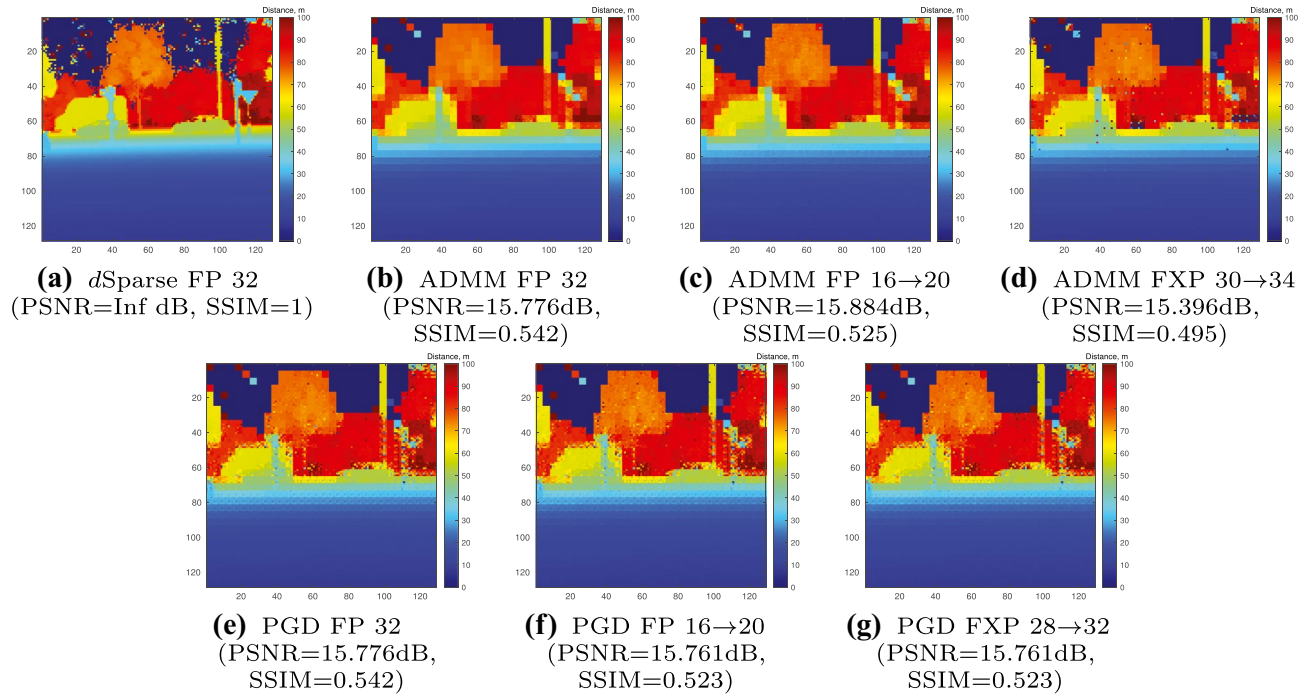


Figure 8 PGD: SSIM v.s. Bitwidth (Short Range Dataset).



(a) *d*Sparse FP 32 (PSNR=Inf dB, SSIM=1)

(b) ADMM FP 32 (PSNR=15.776dB, SSIM=0.542)

(c) ADMM FP 16→20 (PSNR=15.884dB, SSIM=0.525)

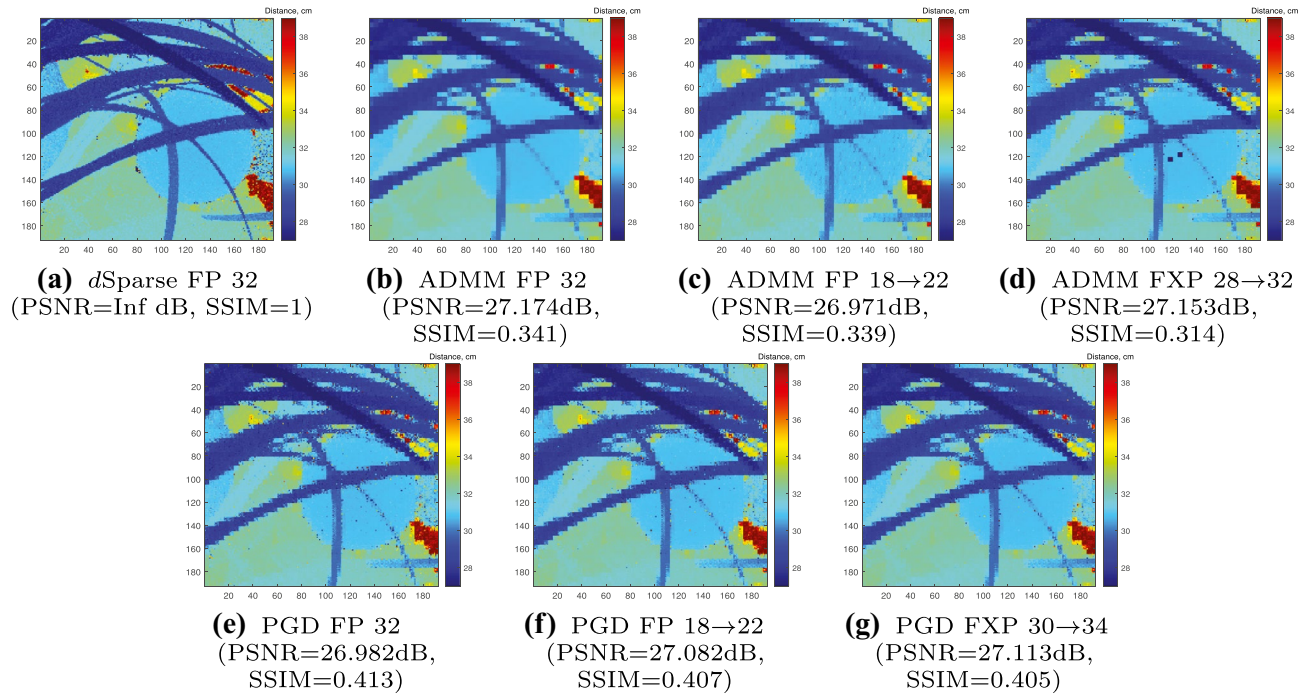
(d) ADMM FXP 30→34 (PSNR=15.396dB, SSIM=0.495)

(e) PGD FP 32 (PSNR=15.776dB, SSIM=0.542)

(f) PGD FP 16→20 (PSNR=15.761dB, SSIM=0.523)

(g) PGD FXP 28→32 (PSNR=15.761dB, SSIM=0.523)

Figure 9 Long range dataset [42]: recovered depth images using **a** *d*Sparse with consistent FP 32 bit; **b** ADMM, consistent FP 32 bit; **c** ADMM, mixed FP 16 to 20 bits; **d** ADMM, mixed FXP 30 to 34 bit; **e** PGD, consistent FP 32 bit; **f** PGD, mixed FP 16 to 20 bits; **g** PGD, mixed FXP 30 to 34 bit.



(a) *d*Sparse FP 32 (PSNR=Inf dB, SSIM=1)

(b) ADMM FP 32 (PSNR=27.174dB, SSIM=0.341)

(c) ADMM FP 18→22 (PSNR=26.971dB, SSIM=0.339)

(d) ADMM FXP 28→32 (PSNR=27.153dB, SSIM=0.314)

(e) PGD FP 32 (PSNR=26.982dB, SSIM=0.413)

(f) PGD FP 18→22 (PSNR=27.082dB, SSIM=0.407)

(g) PGD FXP 30→34 (PSNR=27.113dB, SSIM=0.405)

Figure 10 Underwater dataset A [44]: recovered depth images using **a** *d*Sparse with consistent FP 32 bit; **b** ADMM, consistent FP 32 bit; **c** ADMM, mixed FP 18 to 22 bits; **d** ADMM, mixed FXP 28 to 32 bit; **e** PGD, consistent FP 32 bit; **f** PGD, mixed FP 18 to 22 bits; **g** PGD, mixed FXP 30 to 34 bit.

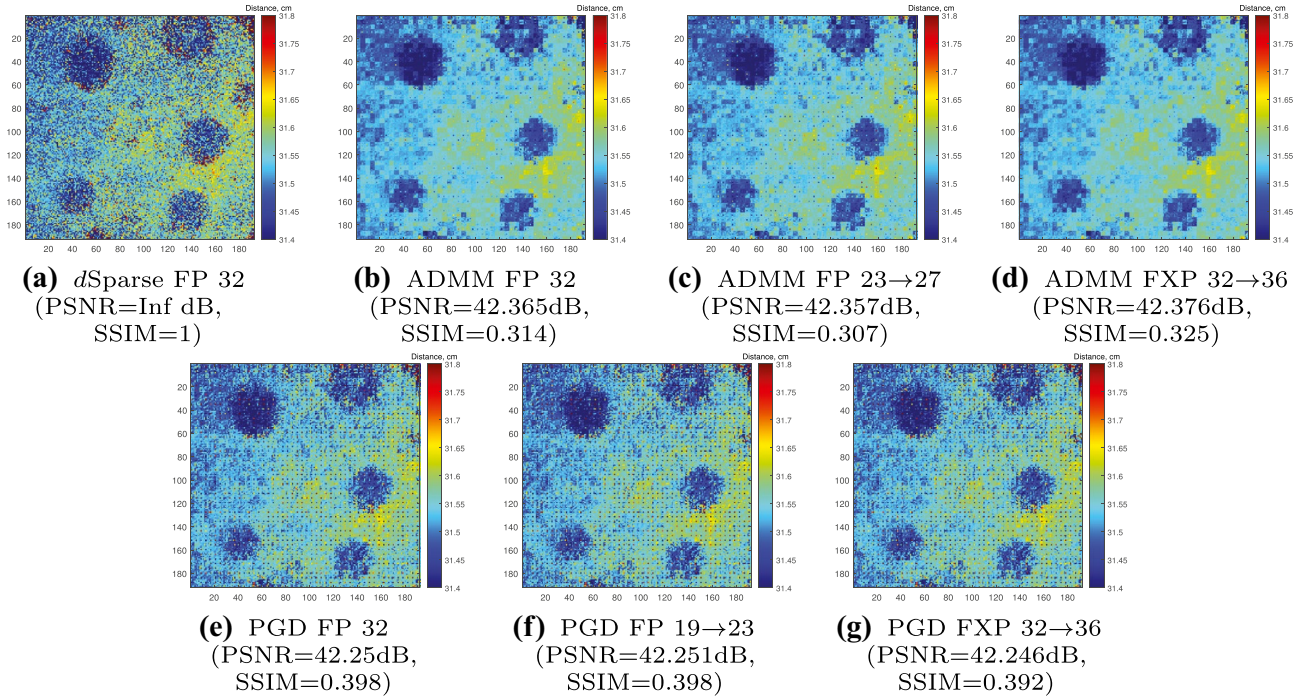


Figure 11 Underwater dataset B [44]: recovered depth images using **a** *d*Sparse with consistent FP 32 bit; **b** ADMM, consistent FP 32 bit; **c** ADMM, mixed FP 23 to 27 bits; **d** ADMM, mixed FXP 32 to 36 bit; **e** PGD, consistent FP 32 bit; **f** PGD, mixed FP 19 to 23 bits; **g** PGD, mixed FXP 32 to 36 bit.

compared to even the highest, fixed precision representations, and is significantly better than the lower precision solutions. This shows a path to even greater resource savings.

iii The use of higher precision in the later iterations of ℓ_1 solvers makes a more significant contribution to the quality of optimization outcome, while the use of lower precision in the earlier iterations is either tolerable or can be well compensated by the later iteration with higher precision.

iv For different mixed precision ℓ_1 solver solutions, PGD incorporates lower precision in general than ADMM to achieve similar fidelity in general, since PGD has simpler computational operations with less opportunity of precision loss.

v Given that we can achieve similar fidelity with mixed precision, we anticipate that significant cost and power savings are achievable with a mixed strategy, when compared computation using fixed precision, whether full or even reduced through all iterations.

Having shown that the use of mixed precision is possible without significant degradation in quality of reconstruction, we now proceed to show how the use of mixed precision can result in further savings in hardware utilization and power consumption in the next section.

4.2 ℓ_1 Solver Accelerators

We have designed and implemented fully pipelined architectures using single and mixed precision with FP and FXP arithmetic on a Xilinx Ultrascale+ ZCU106 architecture using Vivado 2019.1 to fast prototype our design without exhaustive hardware optimization.

Following evaluation of algorithmic performance, the corresponding pipelined architecture has been implemented for either consistent or mixed precision ADMM and PGD to solve the twin lasso optimization problem in Eq. 1, where

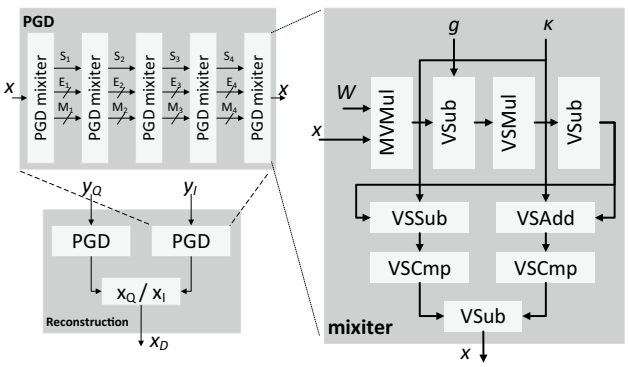


Figure 12 An Example of PGD Architecture.

each iteration corresponds to a pipeline stage with either the same or an allocated, different precision. This strategy can be extended to more iterations, considering mixed precision between batched sub-iterations.

An example of the dataflow architecture of implemented accelerator using PGD ℓ_1 solver with mixed floating point precision is illustrated in Fig. 12. The *MVMul* performs the matrix-vector multiplication, where other *VS* $*$ blocks are vector-scalar addition, subtraction, multiplication and comparison. *VSub* implements the vector subtraction.

The datapath of exponent and mantissa are separated between iterative mixed precision, where the lower bit-width is connected to corresponding part of later higher bit-width pipelines. The outcomes from the twin ℓ_1 solver, x_Q and x_I, adopt a vector element-wise division to obtain the final depth pixels x_D.

Resource utilization is derived from the Xilinx tools using the place and route implementation, while the power is estimated using the Xilinx Power Estimator (XPE). Table 1 presents the resource and power comparisons of both consistent and mixed precision using the ADMM and PGD solvers for all scenarios, which also lists the performance of system reaction latency and processing speed in pixel/s given fixed number of pixel in a depth image. The mixed precision values correspond to the bit width variation chosen according to the depth scaling for each of the scenes while the

Table 1 Resource and quality metrics for consistent and mixed precision. For custom floating point, the bit width = 1+E+M and for custom fixed point, the bit width = 1+I+F.

			Resources			Precision and Performance				
	Arithmetic	Scene Index	LUT	DSP (DSP48E2)	Dynamic Power (W)	Bit Width (bits)	Frequency (MHz)	Latency (ms)	Throughput Pixel/s	Energy (μJ/Pixel)
ℓADMM	FP32	1,2,3,4	20815	110	0.774	(1,8,23)	439	0.26	3.07e5	2.52
$p = 8$	FP27	4	21885	50	0.660	(1,6,20)	407	0.26	3.07e5	2.14
$n = 16$	FP23→27	4	20405	40	0.626	(1,6,16→20)	394	0.24	3.33e5	1.88
	FP22	1,3	18455	25	0.577	(1,6,15)	430	0.27	2.96e5	1.95
	FP18→22	1,3	16192	25	0.535	(1,6,12→15)	430	0.25	3.20e5	1.67
	FP20	2	15680	25	0.541	(1,6,13)	472	0.25	3.20e5	1.69
	FP16→20	2	11092	25	0.494	(1,6,9→13)	473	0.23	3.47e5	1.42
	FXP36	4	9990	60	0.381	(1,23,12)	418	0.09	8.88e5	0.43
	FXP32→36	4	9454	60	0.374	(1,23,8→12)	417	0.09	8.88e5	0.42
	FXP34	2	9375	60	0.375	(1,26,7)	401	0.09	8.88e5	0.42
	FXP30→34	2	8902	57	0.367	(1,25,4→8)	401	0.09	8.88e5	0.41
	FXP32	3	8950	60	0.369	(1,21,10)	406	0.08	1.00e6	0.37
	FXP28→32	3	8461	54	0.357	(1,21,6→10)	406	0.08	1.00e6	0.36
	FXP24	1	6775	30	0.316	(1,13,10)	415	0.07	1.14e6	0.28
	FXP20→24	1	6807	21	0.307	(1,13,6→10)	403	0.07	1.14e6	0.27
PGD	FP32	1,2,3,4	16020	95	0.641	(1,8,23)	415	0.10	8.00e5	0.81
$p = 8$	FP23	4	14845	25	0.537	(1,6,16)	421	0.08	1.00e6	0.54
$n = 16$	FP19→23	4	13116	25	0.510	(1,6,12→16)	419	0.08	1.00e6	0.51
	FP22	3	14115	25	0.524	(1,6,15)	469	0.09	8.88e5	0.59
	FP18→22	3	12368	25	0.497	(1,6,11→15)	453	0.09	8.88e5	0.56
	FP21	1	13070	25	0.506	(1,6,14)	453	0.09	8.88e5	0.57
	FP17→21	1	11663	25	0.484	(1,6,10→14)	453	0.08	1.00e6	0.48
	FP20	2	12020	25	0.496	(1,6,13)	473	0.09	8.88e5	0.56
	FP16→20	2	11092	25	0.473	(1,6,9→13)	469	0.08	1.00e6	0.47
	FXP36	4	7520	80	0.380	(1,23,12)	417	0.01	8.00e6	0.048
	FXP32→36	4	6162	80	0.368	(1,23,8→12)	412	0.01	8.00e6	0.045
	FXP34	3	5860	80	0.366	(1,21,12)	407	0.01	8.00e6	0.045
	FXP30→34	3	5537	80	0.362	(1,21,8→12)	408	0.01	8.00e6	0.044
	FXP32	2	5425	80	0.361	(1,27,4)	428	0.01	8.00e6	0.045
	FXP28→32	2	5060	68	0.329	(1,23→27,4)	400	0.01	8.00e6	0.04
	FXP22	1	5330	20	0.300	(1,12,9)	378	0.008	1.00e7	0.03
	FXP18→22	1	4151	20	0.283	(1,12,5→9)	359	0.008	1.00e7	0.028

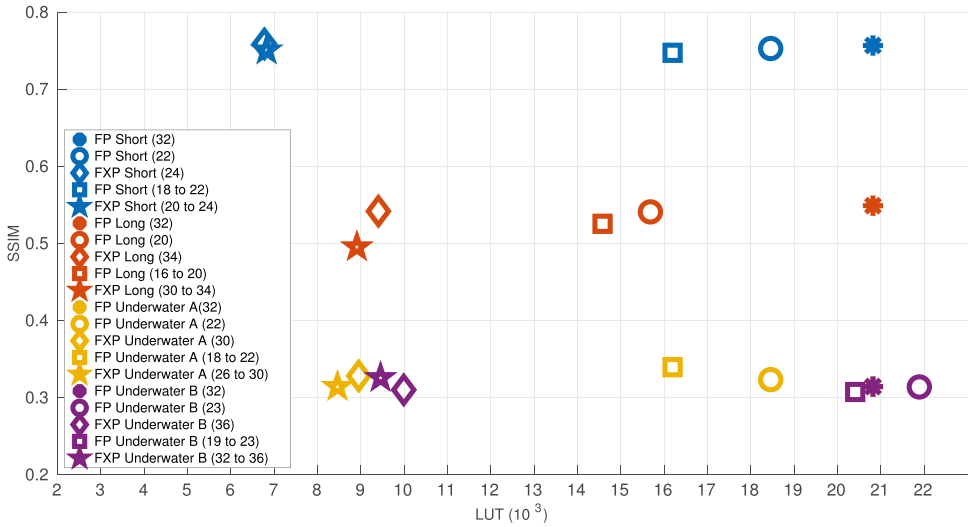
consistent precision cases are those of the highest bit width in the mixed precision combination.

The resource utilization of the Look-Up-Table (LUT) and Digital Signal Processor (DSP) units are reduced by up to 67% and 80% respectively for the ADMM solver using mixed 20 to 24 bit-width fixed-point precision, and by up to 74% and 78% for the PGD solver using mixed 18 to 22 bit-width fixed-point precision. The related power consumption is significantly reduced by up to 60% and 55% for ADMM and PGD solver respectively. Though some mixed fixed-point implementation consume more DSPs than mixed floating-point, they are all less than using single precision floating point.

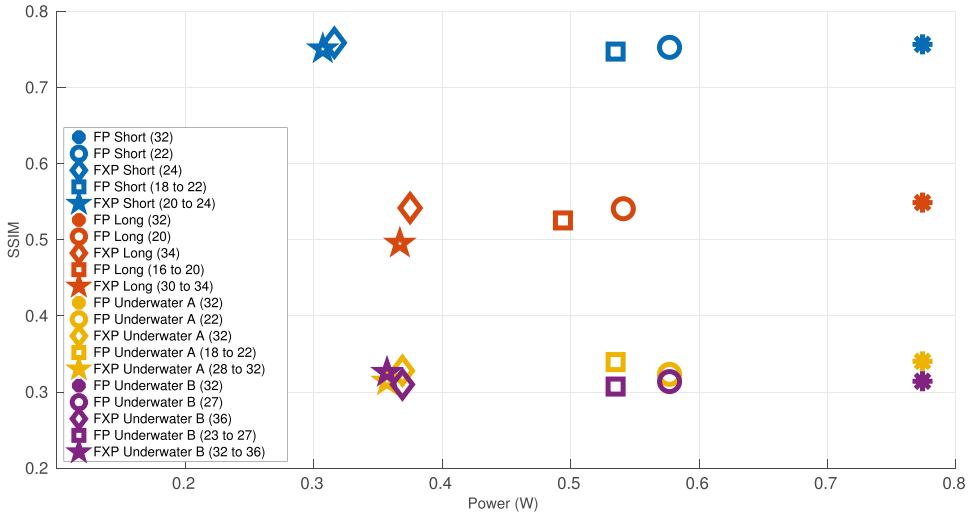
Hence, by adopting mixed precision with either FP or FXP, there are significant hardware resource and power savings when compared to single float precision. Given the similar algorithmic performance between the ADMM and PGD solvers, the outcomes for PGD are better than ADMM in all aspects, both resource utilization and power consumption. Due to the simplicity of the computational operations, the implemented PGD solver gains not only in hardware cost but also in throughput in pixels per second, and in quantified efficiency measured as energy consumption per pixel.

For the hardware implementations, the quality of depth reconstruction is quantified against the consumed resource and power in Figs. 13 and 14 for both the ADMM and PGD solvers. Only SSIM is adopted here as it is more sensitive to individual range errors, though PSNR should have similar trend.

Overall, with an average of 1% degradation in the SSIM metrics for depth reconstruction shown in Figs. 3–11, the

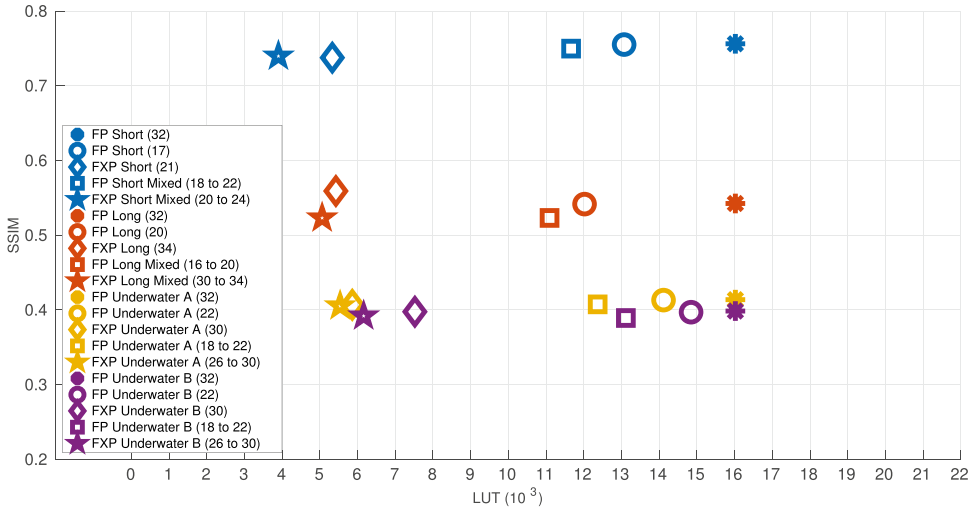


(a) SSIM v.s. LUT

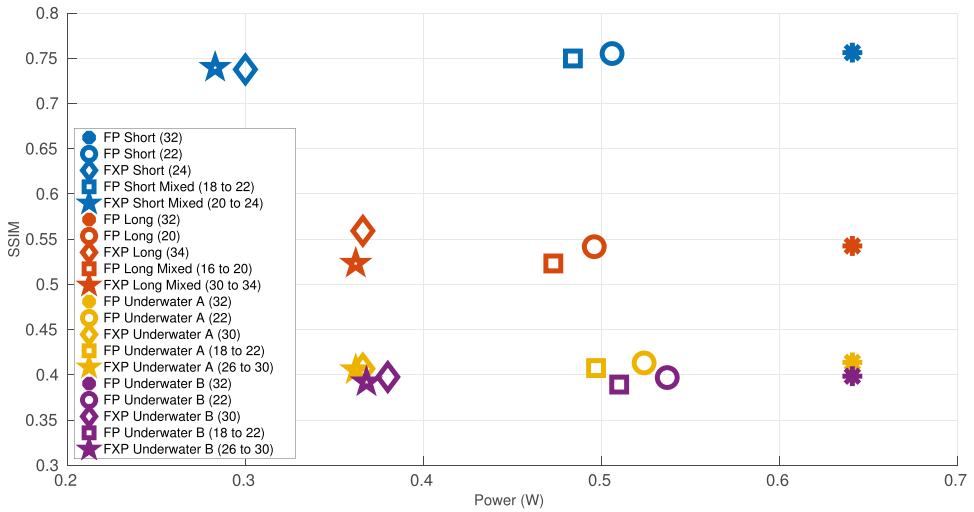


(b) SSIM v.s. Power

Figure 13 SSIM against Resource and Power for the ADMM solver: LUT is considered as the most important resource representing logic area while the power in mW is considered. The higher the SSIM value with less LUT and Power, the better.



(a) SSIM v.s. LUT



(b) SSIM v.s. Power

Figure 14 SSIM against Resource and Power for PGD solver.

cost of the ADMM solver using mixed precision is reduced by up to 67% for Look-Up-Tables (LUT), up to 80% for DSP units, when compared to single precision, and reduced by up to 74% for LUT and 78% for DSP for the PGD solver. The estimated dynamic power consumption using mixed precision is also significantly reduced by over 60% and 55% for the ADMM and PGD solvers respectively compared to single precision, and by approximately 10% compared to the corresponding consistent precision.

For the same sensing scenes with similar SSIM, FXP mixed precision has similar resource utilization to its comparative consistent precision, while the cost with FP mixed precision is slightly lower than its comparative consistent precision with reductions of up to 10% LUT. Accordingly, similar phenomena occur for the power consumption between FXP or FP mixed precision and their comparative consistent precision solutions. The largest advantage of FXP mixed precision over FP occurs in the first short range scenario, which has over 50% reduction in LUT than its corresponding mixed precision accelerator with FP for both the ADMM and PGD solvers.

The bit-width of mixed precision varies between 18 to 28 bits. According to Table 1 and Figs. 3–11, compared to mixed floating-point precision, mixed fixed-point is generally more efficient in terms of power per computation, as measured by μJ/Pixel. In contrast, mixed floating point uses less representation bits, which requires less bandwidth for data communication links. Both benefits are important depending on the design requirements.

Finally, we observe that the implemented ℓ_1 solvers are for processing a single block within the parallel compressive depth reconstruction framework. Hence, the performance

values given in Table 1 are considered as an upper bound of overall real-time depth reconstruction, assuming all the blocks are processed in parallel.

5 Conclusions

In this work, a mixed precision framework using both floating and fixed point arithmetic has been introduced with a pre-computed, static schedule. It has been adopted for compressive depth reconstruction design using approximate convex optimization and linear algebra libraries. By adopting a mixed precision schedule matched to the known stand-off and dynamic range of the LiDAR-sensed scene, incremental precision scaling has been applied to the ℓ_1 solvers, ADMM and PGD, for depth image reconstruction *lasso* using convex optimization. The results show that iterative mixed precision in both floating point and fixed point enables similar performance of depth reconstruction compared to single float precision with significant cost and power reductions using a pipelined architecture; greater than 70% in the best cases.

By analyzing the difference between the ADMM and PGD solvers using either floating point and fixed arithmetic, more efficient depth image reconstruction is enabled with the PGD solver. The benefits of the different arithmetic types have been demonstrated and shown to vary across the different scenarios. In future work, the adaptive runtime strategies to set the mixed precision based on evaluation during the iterative schedule of convex optimization should be explored, according to diverse dynamic sensing ranges and different arithmetic types.

Author Contributions Yun Wu is response for all the research and development in this work as well as the drafting the paper. Andrew M. Wallace, João F.C. Mota, Andreas Aßman are response for providing theoretical and development support as well as editing the draft. Brian Stewart is the project partner with technique support.

Funding This work is supported by the Engineering and Physical Sciences Research Council of the UK (EPSRC) Grant number EP/S000631/1 and the UK MOD University Defence Research Collaboration (UDRC) in Signal Processing.

Data Availability There is no public data available for this work.

Declarations

Ethical Approval There are no human or animal related experiments conducted in this research. All the authors are agreed with this submission. This work is invited extension from previous SiPS 2021 paper entitled "Mixed Precision $\ell 1$ Solver for Compressive Depth Reconstruction: An ADMM Case Study". Our related previous work are in reference [13, 16, 17]. We declare that this work fulfills the standard requirement of invited extended publication with over 30% new materials than its previous conference version. All the outcomes are not published elsewhere at the same time of this submission.

References

1. Park, K., Kim, S., & Sohn, K. (2018). High-precision depth estimation with the 3D lidar and stereo fusion. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 2156–2163). https://doi.org/10.1109/ICRA.2018.8461048
2. Latif, R., & Saddik, A. (2019). Slam algorithms implementation in a UAV, based on a heterogeneous system: A survey. In *2019 4th World Conference on Complex Systems (WCCS)* (pp. 1–6). https://doi.org/10.1109/ICoCS.2019.8930783
3. Qian, L., Wu, J. Y., DiMaio, S. P., Navab, N., & Kazanzides, P. (2020). A review of augmented reality in robotic-assisted surgery. *IEEE Transactions on Medical Robotics and Bionics, 2*(1), 1–16. https://doi.org/10.1109/TMRB.2019.2957061
4. Liu, M., Hou, Z., Sun, Z., Yin, N., Yang, H., Wang, Y., Chu, Z., & Kong, H. (2019) Campus guide: A lidar-based mobile robot. In *2019 European Conference on Mobile Robots (ECMR)* (pp. 1–6). https://doi.org/10.1109/ECMR.2019.8870916
5. Liu, W., Gu, C., O'Neill, M., Qu, G., Montuschi, P., & Lombardi, F. (2020). Security in approximate computing and approximate computing for security: Challenges and opportunities. *Proceedings of the IEEE, 108*(12), 2214–2231. https://doi.org/10.1109/JPROC.2020.3030121
6. Roy, K., & Raghunathan, A. (2015). Approximate computing: An energy-efficient computing technique for error resilient applications. In *2015 IEEE Computer Society Annual Symposium on VLSI* (pp. 473–475). https://doi.org/10.1109/ISVLSI.2015.130
7. Pandey, P., He, Q., Pompili, D., & Tron, R. (2018). Light-weight object detection and decision making via approximate computing in resource-constrained mobile robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 6776–6781). https://doi.org/10.1109/IROS.2018.8594200
8. Ibrahim, A., Osta, M., Alameh, M., Saleh, M., Chible, H., & Valle, M. (2018) Approximate computing methods for embedded machine learning. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)* (pp. 845–848). https://doi.org/10.1109/ICECS.2018.8617877
9. Agrawal, A., Choi, J., Gopalakrishnan, K., Gupta, S., Nair, R., Oh, J., Prener, D.A., Shukla, S., Srinivasan, V., & Sura, Z. (2016) Approximate computing: Challenges and opportunities. *2016 IEEE International Conference on Rebooting Computing (ICRC)* 1–8.
10. Webster, E. A. G., Grant, L. A., & Henderson, R. K. (2012). A high-performance single-photon avalanche diode in 130-nm CMOS imaging technology. *IEEE Electron Device Letters, 33*(11), 1589–1591. https://doi.org/10.1109/LED.2012.2214760
11. Aßmann, A., Stewart, B., Mota, J. F. C., & Wallace, A. M. (2019) Compressive super-pixel lidar for high-framerate 3D depth imaging. In *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)* (pp. 1–5). https://doi.org/10.1109/GlobalSIP45357.2019.8969177

12. Nguyen, M. U., Dao, T. T., & Tang, V. H. (2018). Efficient depth image reconstruction using accelerated proximal gradient method. In *2018 10th International Conference on Knowledge and Systems Engineering (KSE)* (pp. 1–6). https://doi.org/10.1109/KSE.2018.8573361
13. Aßmann A., Wu, Y., Stewart, B., & Wallace, A. M. (2021). Accelerated 3D image reconstruction for resource constrained systems. In *2020 28th European Signal Processing Conference (EUSIPCO)* (pp. 565–569). https://doi.org/10.23919/Eusipco47968.2020.9287749
14. Gürel, N. M., Kara, K., Stojanov, A., Smith, T., Lemmin, T., Alistarh, D., Püschel, M., & Zhang, C. (2020). Compressive sensing using iterative hard thresholding with low precision data representation: Theory and applications. *IEEE Transactions on Signal Processing, 68,* 4268–4282. https://doi.org/10.1109/TSP.2020.3010355
15. Wills, A., Mills, A., & Ninness, B. (2011). FPGA implementation of an interior-point solution for linear model predictive control. *18th IFAC World Congress*.
16. Wu, Y., Mota, J. F. C., & Wallace, A. M. (2020). Approximate lasso model predictive control for resource constrained systems. In *2020 Sensor Signal Processing for Defence Conference (SSPD)* (pp. 1–5). https://doi.org/10.1109/SSPD47486.2020.9272000
17. Wu, Y., Assmann, A., Stewart, B., & Wallace, A. M. (2021). Energy efficient approximate 3d image reconstruction. *IEEE Transactions on Emerging Topics in Computing*, *1*. https://doi.org/10.1109/TETC.2021.3116471
18. Abdelfattah, A., Anzt, H., Boman, E. G., Carson, E., Cojean, T., Dongarra, J., Fox, A., Gates, M., Higham, N. J., Li, X. S., Loe, J., Luszczek, P., Pranesh, S., Rajamanickam, S., Ribizel, T., Smith, B. F., Swirydowicz, K., Thomas, S., Tomov, S., Tsai, Y. M., & Yang, U. M. (2021). A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. *The International Journal of High Performance Computing Applications, 35*(4), 344–369. https://doi.org/10.1177/10943420211003313
19. Carson, E., & Higham, N. (2018). Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM Journal on Scientific Computing, 40,* 817–847. https://doi.org/10.1137/17M1140819
20. Hameed, R., Qadeer, W., Wachs, M., Azizi, O., Solomatnikov, A., Lee, B. C., Richardson, S., Kozyrakis, C., & Horowitz, M. (2010). Understanding sources of inefficiency in general-purpose chips. *SIGARCH Computer Architecture News, 38*(3), 37–47. https://doi.org/10.1145/1816038.1815968
21. Hernández-Marín, S., Wallace, A. M., & Gibson, G. J. (2007). Bayesian analysis of lidar signals with multiple returns. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 29*(12), 2170–2180. https://doi.org/10.1109/TPAMI.2007.1122
22. Wallace, A. M., Ye, J., Krichel, N. J., McCarthy, A., Collins, R. J., & Buller, G. S. (2010). Full waveform analysis for long-range 3D imaging laser radar. *EURASIP Journal on Advances in Signal Processing*. https://doi.org/10.1155/2010/896708
23. Halimi, A., Tobin, R., McCarthy, A., McLaughlin, S., & Buller, G. S. (2017) Restoration of multilayered single-photon 3D Lidar images. In *25th IEEE European Signal Processing Conference (EUSIPCO)* (pp. 708–712). https://doi.org/10.23919/EUSIPCO.2017.8081299
24. Tachella, J., Altmann, Y., Mellado, N., McCarthy, A., Tobin, R., Buller, G. S., Tourneret, J.-Y., & McLaughlin, S. (2019). Real-time 3D reconstruction from single-photon lidar data using plug-and-play point cloud denoisers. *Nature Communications, 10*(1), 4984. https://doi.org/10.1038/s41467-019-12943-7
25. Patanwala, S. M., Gyongy, I., Dutton, N. A. W., Rae, B. R., & Henderson, R .K. (2019). A reconfigurable 40nm CMOS SPAD array for lidar receiver validation. In *International Image Sensor Workshop (IISW)*.
26. Henderson, R. K., Johnston, N., Hutchings, S. W., Gyongy, I., Abbas, T. A., Dutton, N., Tyler, M., Chan, S., & Leach, J. (2019) 5.7 a 256×256 40nm/90nm CMOS 3D-stacked 120db dynamic-range reconfigurable time-resolved SPAD imager. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)* (pp. 106–108). https://doi.org/10.1109/ISSCC.2019.8662355
27. Candès, E., & Romberg, J. (2007). Sparsity and incoherence in compressive sampling. *Inverse Problems, 23*(3), 969–985. https://doi.org/10.1088/0266-5611/23/3/008
28. Donoho, D. L. (2006). Compressed sensing. *IEEE Transactions on Information Theory, 52*(4), 1289–1306. https://doi.org/10.1109/TIT.2006.871582
29. Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological) 58*, 267–288. https://doi.org/10.2307/2346178
30. Boyd, S., Parikh, N., Chu, E., & Peleato, B. (2011) Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning, 3*(1), 1–122 arXiv:0307085 [cond-mat]. https://doi.org/10.1561/2200000016
31. Parikh, N., & Boyd, S. (2014). *Proximal algorithms, 1*(3), 127–239. https://doi.org/10.1561/2400000003
32. Flegar, G., Scheidegger, F., Novaković, V., Mariani, G., Tom's, A. E., Malossi, A. C. I., & Quintana-Ortí, E. S. (2019). Floatx: A c++ library for customized floating-point arithmetic. *ACM Transactions on Mathematical Software (TOMS), 45*(4). https://doi.org/10.1145/3368086
33. Muller, J.-M., Brunie, N., de Dinechin, F., Jeannerod, C.-P., Joldes, M., Lefvre, V., Melquiond, G., Revol, N., & Torres, S. (2018). Handbook of floating-point arithmetic. Birkhäuser. https://doi.org/10.1007/978-0-8176-4705-6
34. Langou, J., Langou, J., Luszczek, P., Kurzak, J., Buttari, A., & Dongarra, J. (2006) Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. SC '06* (p. 113). Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/1188455.1188573
35. Yu, H., Han, Q., Li, J., Shi, J., Cheng, G.-L., & Fan, B. (2020) Search what you want: Barrier panelty NAS for mixed precision quantization. ArXiv abs/2007.10026.
36. Smith, S. W. (1997). *The scientist and engineer's guide to digital signal processing*. California Technical Publishing, USA.
37. Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press. https://doi.org/10.1017/CBO9780511804441
38. Beck, A., & Teboulle, M. (2009). *Gradient-based algorithms with applications to signal-recovery problems*. Cambridge University Press. https://doi.org/10.1017/CBO9780511804458.003
39. Xilinx. (2020). Vivado design suite user guide: High-level synthesis. Accessed on 11/10/2020 https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug902-vivado-high-level-synthesis.pdf
40. Silberman, N., Hoiem, D., Kohli, P., & Fergus, R. (2012). Indoor segmentation and support inference from RGBD images. In A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, & C. Schmid (Eds.), *Computer Vision – ECCV 2012* (pp. 746–760). Springer, Berlin, Heidelberg.
41. Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004) Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing, 13*(4), 600–612. https://doi.org/10.1109/TIP.2003.819861
42. Gaidon, A., Wang, Q., Cabon, Y., & Vig, E. (2016). Virtualworlds as proxy for multi-object tracking analysis. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 4340–4349). IEEE Computer Society, Los Alamitos, CA, USA. https://doi.org/10.1109/CVPR.2016.470
43. Chhabra, P., Maccarone, A., McCarthy, A., Buller, G., & Wallace, A. (2016). Discriminating underwater LiDAR target signatures

using sparse multi-spectral depth codes. In *2016 Sensor Signal Processing for Defence, SSPD 2016*. https://doi.org/10.1109/SSPD.2016.7590595

44. Chhabra, P., Maccarone, A., McCarthy, A., Buller, G., & Wallace, A. (2016). Discriminating underwater lidar target signatures using sparse multi-spectral depth codes. In *2016 Sensor Signal Processing for Defence (SSPD)* (pp. 1–5). https://doi.org/10.1109/SSPD.2016.7590595

Yun Wu received the B.Sc. degree in electronic and information engineering from Dalian Nationalities University, Dalian, China, in 2003, the M.Sc. degree in circuits and system from Hunan University, Changsha, China, in 2007, the M.Sc. degree in radio frequency communication systems from the University of Southampton, Southampton, U.K., in 2008, and the Ph.D. degree in electronic engineering from Queen's University Belfast, Belfast, U.K., in 2014. He is currently associate professor and UDRC research associate with Heriot-Watt University. His research interests include wireless/image signal processing, reconfigurable architecture synthesis, and approximate computing.

Andrew M. Wallace received his BSc and PhD degrees from the University of Edinburgh in 1972 and 1975 respectively. He is an Emeritus Professor of Signal and Image Processing at Heriot-Watt University. His research interests include LiDAR and 3D vision, image and signal processing, and accelerated computing. He has published extensively and has secured funding from EPSRC, the EU and other sponsors. He is a Chartered Engineer and a Fellow of the Institute of Engineering Technology.

João F. C. Mota received the M.Sc. degree and the Ph.D. degree in Electrical and Computer Engineering from the Technical University of Lisbon, Portugal, in 2008 and 2013, respectively. He also received the Ph.D. degree in Electrical and Computer Engineering from Carnegie Mellon University, PA, USA, in 2013.

From 2013 to 2016, he was Senior Research Associate at University College London, London, U.K. In 2017, he became Assistant Professor in the School of Engineering and Physical Sciences at Heriot-Watt University, Edinburgh, U.K., where he is also affiliated with the Institute of Sensors, Signals, and Systems. His current research interests include theoretical and practical aspects of high-dimensional data processing, inverse problems, optimization theory, machine learning, data science, and distributed information processing and control. He was the recipient of the 2015 IEEE Signal Processing Society Young Author Best Paper Award for the paper "Distributed Basis Pursuit", published in IEEE Transactions on Signal Processing.

Andreas Aßmann 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 at Heriot-Watt University, Edinburgh in 2021. Since November 2020 he is a System Architect with STMicroelectronics R&D Ltd. in Edinburgh, Scotland. His research interests include efficient signal processing of active imaging systems and computer vision.

Brian D. Stewart received his BSc in Electronics and PhD in Computer Vision from Dundee University in 1989 and 1992 respectively. During his time at STMicroelectronics he has focused on software development, modelling and image/signal processing and is a Design Architect within Imaging Division in Edinburgh, Scotland.