A Distributed Plug-and-Play MCMC Algorithm for High-Dimensional Inverse Problems

Maxime Bouton, Pierre-Antoine Thouvenin, Audrey Repetti and Pierre Chainais

***Abstract*—Markov Chain Monte Carlo (MCMC) algorithms are standard approaches to solve imaging inverse problems and quantify estimation uncertainties, a key requirement in absence of ground-truth data. To improve estimation quality, Plug-and-Play MCMC algorithms, such as PnP-ULA, have been recently developed to accommodate priors encoded by a denoising neural network. Designing scalable samplers for high-dimensional imaging inverse problems remains a challenge: drawing and storing high-dimensional samples can be prohibitive, especially for high-resolution images. To address this issue, this work proposes a distributed sampler based on approximate data augmentation and PnP-ULA to solve very large problems. The proposed sampler uses lightweight denoising convolutional neural network, to efficiently exploit multiple GPUs on a Single Program Multiple Data architecture. Reconstruction performance and scalability are evaluated on several imaging problems. Communication and computation overheads due to the denoiser are carefully discussed. The proposed distributed approach noticeably combines three very precious qualities: it is scalable, enables uncertainty quantification, for a reconstruction performance comparable to other PnP methods.**

***Index Terms*—Markov chain Monte Carlo algorithms, Langevin algorithm, Plug-and-Play prior, distributed computing**

I. Introduction

This work focuses on high-dimensional imaging inverse problems, aimed at recovering an unknown image $\overline{\boldsymbol{x}} \in \mathbb{R}^N$ from degraded observations $\boldsymbol{y} \in \mathbb{R}^M$, linked by a model

$$\boldsymbol{y} = \mathcal{A}(\overline{\boldsymbol{x}}), \tag{1}$$

where $\mathcal{A} : \mathbb{R}^N \to \mathbb{R}^M$ models both the deterministic degradations and the noise affecting $\overline{\boldsymbol{x}}$. Both M and N can range from 10^6 to 10^{10}, raising computational challenges related to prohibitive memory requirements and runtime. Bayesian inference is a usual approach to account for uncertainties, based on the posterior distribution combining the likelihood associated with (1) with prior information on $\overline{\boldsymbol{x}}$ [1]. In this paper, we consider problems whose posterior distribution is described by a probability density function (pdf) of the form

$$\pi(\boldsymbol{x}|\boldsymbol{y}) \propto p(\boldsymbol{x}) \exp\big(-f_1(\boldsymbol{H}_1\boldsymbol{x}) - f_2(\boldsymbol{H}_2\boldsymbol{x})\big), \tag{2}$$

where $f_1 \colon \mathbb{R}^{M_1} \to (-\infty, +\infty]$ is a proper, Lipschitz-differentiable function, $\boldsymbol{H}_1 \colon \mathbb{R}^N \to \mathbb{R}^{M_1}$ and $\boldsymbol{H}_2 \colon \mathbb{R}^N \to \mathbb{R}^{M_2}$ are linear operators, and $f_2 \colon \mathbb{R}^{M_2} \to (-\infty, +\infty]$ is a proper, lower semi-continuous (l.s.c) convex function. The function $p \colon \mathbb{R}^N \to [0, 1]$ can represent a pdf corresponding to part of the prior distribution, based for instance on a neural network (NN) [2], [3], [4]. Conditions to ensure the existence of a proper prior with density p have for instance been investigated in [3] for a denoising NN. The dependence on $\boldsymbol{y}$ is omitted in (2), so that $f_1 \circ \boldsymbol{H}_1$ and $f_2 \circ \boldsymbol{H}_2$ can come from either the likelihood or the prior distributions.

Solving (1) in a high-dimensional setting is challenging, especially when quantifying uncertainties, which is critical for decision-making processes, and when calibrated data is hard or impossible to obtain (*e.g.*, in astronomy [5]). Markov-chain Monte Carlo (MCMC) algorithms are commonly used to explore the posterior distribution (2), allowing estimates to be formed with credibility intervals to quantify uncertainties. Nevertheless, they usually do not scale well. Different approaches have been proposed in the recent years to design scalable MCMC algorithms. For instance, samplers inspired by primal-dual proximal algorithms [6] have been proposed in [7], [8]. An approximate data augmentation approach similar to half-quadratic splitting strategy [9] has been developed in [10], addressed with a Gibbs samplers referred to as Split Gibbs sampler (SGS). A distributed sampler based on [10] has been proposed in [11], exploiting the hypergraph structure underlying the approximate posterior distribution (2) to partition both $\boldsymbol{y}$ and $\boldsymbol{x}$ over several workers. This enables Proximal Stochastic Gradient Langevin Algorithms (PSGLA) [12] transitions to be implemented within SGS on a single program multiple data (SPMD) architecture [13]. In this setting, each worker conducts the same operations on disjoint subsets of $\boldsymbol{y}$ and $\boldsymbol{x}$, with a small number of communications whenever the operations involved are localized. Compared to a client-server implementation, this approach was shown to offer more flexibility to (i) use multiple workers with a balanced computing load, and (ii) limit communication costs [11].

Finally, MCMC methods have recently been paired with data-driven priors through the use of NNs to achieve high-quality estimation. Learned priors encoded by NNs have emerged as powerful tools to incorporate data-driven knowledge [14]. In particular, Plug-and-Play (PnP) approaches [15], [16] leverage a generic denoiser within a standard inference algorithm. Several PnP MCMC algorithms have been proposed using different families of learned priors [3], [17], [18], [19], [4]. However, none of these algorithms has been designed with a strong focus on scalability and distributed computing.

This work is aimed at designing a distributed data-driven MCMC algorithm to achieve state-of-the-art reconstruction quality, while enabling uncertainty quantification in high di-

This work was supported by the ANR project "Chaire IA Sherlock" ANR-20-CHIA-0031-01 hold by P. Chainais, the national support within the *programme d'investissements d'avenir* ANR-16-IDEX-0004 ULNE, Région HDF, and the CNRS IEA "DAISHI" hold by P.-A. Thouvenin and A. Repetti. The work of A. Repetti was partly supported by the EPSRC grant EP/X028860 and a grant from the Simons Foundation within the Isaac Newton Programme. HPC and storage resources were provided by GENCI at IDRIS on the supercomputer Jean Zay's V100 partition *via* the grant 2024-AD010615597.

mensional settings. Our main contribution is to propose a multi-GPU distributed MCMC algorithm to address problems, whose size exceeds the memory of a single GPU. The proposed algorithm relies on the structure of a recent distributed Gibbs sampler [11] designed for imaging inverse problems, which is statistically equivalent to its serial counterpart. Specifically, our approach relies on the design of a distributed framework for the NN encoding the prior used in the sampler. In this context, we focus on fairly light NN architectures, as the number of parameters of the NN can have a notable memory cost for distributed computing.

The main classes of denoisers used in PnP-MCMC algorithms are convolutional NN (CNN) [20], Normalizing Flows (NF) [21] and Denoising Diffusion Models (DDM) [22]. Both NF and DDM are usually encoded by a very large number of parameters and involve non-local operations, which precludes an efficient distributed implementation. In contrast, CNNs offer promising opportunities for distributed computing. Each of their layers only involve convolution operators and, in general, entry-wise non-linearities. The former can usually be distributed efficiently, whereas the latter are embarrassingly parallel. Besides, CNNs designed using unrolled algorithms [23], such as the Deep Dual Forward-Backward (DDFB) network [24], [25], have been shown to offer a good compromise between restoration performance and reduction in the number of parameters compared to other alternatives.

Inspired by [10], [11], we propose a distributed sampler based on the PnP unadjusted Langevin Algorithm (PnP-ULA) [3], using CNNs as pre-trained denoisers. In a preliminary work [26], we presented simulation results suggesting that a distributed version of PnP-ULA based on DDFB allows for significant scalability and acceleration gains. In the current work, we significantly extend these results to general CNNs, describing (i) the distributed approach adopted for all the operators involved in the proposed sampler (including the CNNs), and (ii) the computation and communication overheads induced by different choices of CNN denoisers, namely DnCNN [27], DRUNet [16] and DDFB. We also provide extensive simulation results to validate the scalability and reconstruction quality of the proposed sampler.

The paper is organized as follows. Section II formulates the problem and introduces the proposed sampler. Locality assumptions for an efficient distributed implementation are introduced in section II-A. Section II-B discusses denoiser structures favorable to a distributed computing setting, and describes the proposed distributed implementation on an SPMD architecture. Section II-C exploits the localized structure of the denoiser to design the proposed distributed PnP-MCMC algorithm. Section III presents the experiment settings used to evaluate the proposed algorithm in terms of reconstruction quality and scalability. Scalability potential is discussed for different representative choices of denoisers in terms of memory costs, communication costs and runtime. Experiment results are reported and discussed in Section IV. Section V gathers conclusions and research perspectives.

Notation. The union and disjoint union of sets are denoted by $\cup$ and $\sqcup$, respectively. The cardinality of a set is denoted by $\sharp$. The adjoint of $\boldsymbol{H} \in \mathbb{R}^{M\times N}$ is denoted by $\boldsymbol{H}^*$, and $\mathbf{1}_N \in \mathbb{R}^N$ is the unit vector with all coefficients equal to 1. The indicator function of a non-empty set $\mathcal{C} \subset \mathbb{R}^N$ is denoted by $\iota_{\mathcal{C}}$, with $\iota_{\mathcal{C}}(\boldsymbol{x}) = 0$ if $\boldsymbol{x} \in \mathcal{C}$, and $+\infty$ otherwise, and we further use the notation $\mathbb{1}_{\mathcal{C}} : \boldsymbol{x} \mapsto e^{-\iota_{\mathcal{C}}(\boldsymbol{x})}$. The proximity operator [28] of $\varphi\colon \mathbb{R}^N \to (-\infty,+\infty]$ at $\boldsymbol{x} = (x^{(n)})_{1\le n\le N} \in \mathbb{R}^N$ is defined as $\operatorname{prox}_{\varphi}(\boldsymbol{x}) = \operatorname{argmin}_{\boldsymbol{u}\in\mathbb{R}^N} \left\{\varphi(\boldsymbol{u}) + \frac{1}{2}\|\boldsymbol{u}-\boldsymbol{x}\|_2^2\right\}$. In particular, $\operatorname{prox}_{\iota_{\mathcal{C}}}$ is the orthogonal projector $\operatorname{proj}_{\mathcal{C}}$ onto $\mathcal{C}$.

II. Proposed approach

This section describes the proposed approach, building on SGS [10] and PnP-ULA [3]. Section II-A introduces the model considered, specifying the operator locality assumptions required for an efficient distributed sampling strategy. Section II-B discusses several CNNs amenable to an efficient distributed implementation. Section II-C finally introduces the proposed distributed sampler.

A. Problem statement

Operator locality. We consider the case when operators in (2) only act locally with respect to a partition of $\{1,\dots,N\}$. Then, the algorithm can be split into loosely-dependent tasks, that can be carried out by different workers. Dependencies between the tasks are handled by lightweight communications between the workers. Operator locality is formalized below.

Definition 1 (Local selection operators) Let $(\mathbb{V}_b)_{1\le b\le B}$ be a partition of $\{1,\dots,N\}$ into $B>1$ subsets, and, for every $b \in \{1,\dots,B\}$, $\sharp\,\mathbb{V}_b = N_b \neq 0$. Let $(\widetilde{\mathbb{V}}_b)_{1\le b\le B}$ be an extension of this partition, such that, for every $b \in \{1,\dots,B\}$, $\mathbb{V}_b \subseteq \widetilde{\mathbb{V}}_b$ and $\sharp\,\widetilde{\mathbb{V}}_b = \widetilde{N}_b > N_b$. Let, for every $b \in \{1,\dots,B\}$, $\boldsymbol{S}_b \in \{0,1\}^{\widetilde{N}_b\times N}$ be a selection operator (*i.e.*, each row only contains 0 but a single 1) satisfying

$$\widetilde{\mathbb{V}}_b := \bigcup_{j\in\{1,\dots,\widetilde{N}_b\}} \{n \in \{1,\dots,N\} \,|\, \boldsymbol{S}_b^{(j,n)} = 1\}. \tag{3}$$

Then, $\{\boldsymbol{S}_b\}_{1\le b\le B}$ is called a *family of local selection operators* with respect to $(\mathbb{V}_b)_{1\le b\le B}$ if

(i) for every $b \in \{1,\dots,B\}$, $\widetilde{N}_b - N_b < \min_{b'\in\{1,\dots,B\}} N_{b'}$;
(ii) $\max_{n\in\{1,\dots N\}} \sharp\{b \in \{1,\dots,B\} \,|\, n \in \widetilde{\mathbb{V}}_b\} < B$.

A few remarks can be done on Definition 1. First, it implies that $\{1,\dots,N\} = \bigsqcup_{b=1}^B \mathbb{V}_b = \bigcup_{b=1}^B \widetilde{\mathbb{V}}_b$. Further, condition (ii) ensures that each element $n \in \{1,\dots,N\}$ is only selected by a few operators $\{\boldsymbol{S}_b\}_{1\le b\le B}$. Combined with (i), it further controls the size of the overlapping areas. In other words, the combination of conditions (i) and (ii) ensures that $(\widetilde{\mathbb{V}}_b)_{1\le b\le B}$ is close to be a partition of $\{1,\dots,N\}$. In a distributed computing setting, overlapping areas define the messages exchanged between the workers and in practice, the smaller $\widetilde{N}_b - N_b$, the better. Hence, to limit communication costs, we will consider *localized operators* as defined below.

Definition 2 (Localized operator) An operator $G\colon \mathbb{R}^N \to \mathbb{R}^M$ is said to be *localized* if it can be decomposed into a family of local selection operators $(\boldsymbol{S}_b)_{1\le b\le B}$ as

$$(\forall b \in \{1,\dots,B\})(\forall \boldsymbol{x} \in \mathbb{R}^N) \quad ([G(\boldsymbol{x})]^{(i)})_{i\in\mathbb{I}_b} = G_b(\boldsymbol{S}_b\boldsymbol{x}), \tag{4}$$

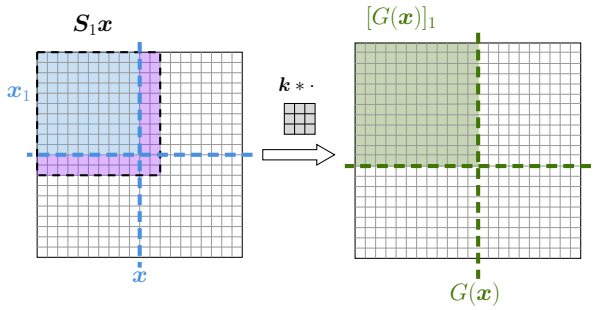

Fig. 1. Illustration of a 2D linear convolution operator G defined by a 3×3 kernel $\boldsymbol{k}$, applied to an image $\boldsymbol{x}$ of size $N = 20 \times 20$ (hence $G(\boldsymbol{x}) \in \mathbb{R}^M$ with $M = 22 \times 22$). The input $\boldsymbol{x}$ is partitioned into $B = 4$ blocks (in dashed blue lines). Resulting blocks in $G(\boldsymbol{x})$ are delimited by dashed green lines. The block $\boldsymbol{x}_1$ assigned to worker 1 is in blue, with corresponding block $[G(\boldsymbol{x})]_1$ in green. The purple area contains pixels received by worker 1 to compute $[G(\boldsymbol{x})]_1$. Pixels in $\boldsymbol{S}_1\boldsymbol{x}$ are in the union of the blue and purple areas.

where $(\mathbb{I}_b)_{1\le b\le B}$ is a partition of $\{1,\dots,M\}$ into B subsets, with $M_b = \sharp\mathbb{I}_b > 0$, and $G_b\colon \mathbb{R}^{\widetilde{N}_b} \to \mathbb{R}^{M_b}$. To simplify the notation, we use hereafter $[G(\boldsymbol{x})]_b$ instead of $([G(\boldsymbol{x})]^{(i)})_{i\in\mathbb{I}_b}$.

A localized operator $G\colon \mathbb{R}^N \to \mathbb{R}^M$ with respect to a family of local selection operators $(\boldsymbol{S}_b)_{1\le b\le B}$ can be evaluated in a distributed manner (see, *e.g.*, [11] for details). Let $\boldsymbol{x} \in \mathbb{R}^N$ and $b \in \{1,\dots,B\}$. As per (4), each block $[G(\boldsymbol{x})]_b$ only depends on the locally selected block $\boldsymbol{S}_b\boldsymbol{x}$ (or equivalently, $\mathbb{I}_b$ only depends on the neighborhood $\widetilde{\mathbb{V}}_b$ as given in Definition 1). In a distributed computing setting, computations underlying $[G(\boldsymbol{x})]_b$ can be assigned to a single worker b, with $(x^{(n)})_{n\in\mathbb{V}_n}$ directly stored in its local memory. To compute $[G(\boldsymbol{x})]_b$, the worker b receives $(x^{(n)})_{n\in\widetilde{\mathbb{V}}_b\setminus\mathbb{V}_b}$ from a small number of workers $b' \neq b$, each sending messages of small size. The choice of the partition $(\mathbb{V}_b)_{1\le b\le B}$ directly impacts the local selection operators underlying the operator G, hence significantly affects the efficiency of the associated communications.

Example 1 Figure 1 illustrates Definitions 1 and 2 with a 2D linear convolution operator defined by a kernel $\boldsymbol{k}$ of size 3×3. An input image $\boldsymbol{x} \in \mathbb{R}^N$ of size $N = 20 \times 20$ is considered, leading to $G(\boldsymbol{x}) \in \mathbb{R}^M$ with $M = 22\times 22$. Dashed blue lines represent the partition of $\boldsymbol{x}$ into $B = 4$ blocks $\boldsymbol{x}_b = (x^{(n)})_{n\in\mathbb{V}_b}$, with $b \in \{1,\dots,4\}$. Each block is assigned to a single worker. The partition $[G(\boldsymbol{x})]_b = \left([G(\boldsymbol{x})]^{(i)}\right)_{i\in\mathbb{I}_b}$ is indicated with dashed green lines. The block $\boldsymbol{x}_1$ (in blue) is assigned to worker 1, as well as the corresponding output block $[G(\boldsymbol{x})]_1$ (in green). The pixels $(x^{(n)})_{n\in\widetilde{\mathbb{V}}_1\setminus\mathbb{V}_1}$ (in purple) are required by worker 1 to compute $[G(\boldsymbol{x})]_1$. These pixels form a *ghost region* [29, Section 7.5.2], and are received from other workers before computations can be conducted on worker 1.

Model assumptions. Let $B \in \mathbb{N}^*$ be the number of workers available. The posterior distribution (2) is assumed to satisfy the following assumptions, with associated notation in Table I.

Assumption 1

(A1) The composition $f_1 \circ \boldsymbol{H}_1\colon \mathbb{R}^N \to (-\infty,+\infty]$ is Lipschitz-differentiable, with Lipschitz constant $L > 0$.

(A2) $f_2\colon \mathbb{R}^{M_2} \to \left(-\infty,+\infty\right]$ is proper, l.s.c. and convex.

TABLE I
NOTATION FOR LOCALIZED OPERATORS $\boldsymbol{H}_i$, WITH $i \in \{1,2\}$.

$\boldsymbol{H}_i$	Linear operator from $\mathbb{R}^N$ to $\mathbb{R}^{M_i}$;
$(\boldsymbol{H}_{i,b})_{1\le b\le B}$	Decomposition of $\boldsymbol{H}_i$ with $\boldsymbol{H}_{i,b}\colon \mathbb{R}^{\widetilde{N}_{i,b}} \to \mathbb{R}^{M_{i,b}}$ and $[\boldsymbol{H}_i\boldsymbol{x}]_b = \boldsymbol{H}_{i,b}\boldsymbol{S}_{i,b}\boldsymbol{x}$;
$(\boldsymbol{S}_{i,b})_{1\le b\le B}$	Family of local selection operators;
$(\mathbb{V}_b)_{1\le b\le B}$	Partition of $\{1,\dots,N\}$ (same for both $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$);
$(N_b)_{1\le b\le B}$	Cardinality for $(\mathbb{V}_b)_{1\le b\le B}$, with $\sum_{b=1}^B N_b = N$;
$(\widetilde{\mathbb{V}}_{i,b})_{1\le b\le B}$	Extension of $(\mathbb{V}_b)_{1\le b\le B}$ with overlapping;
$(\widetilde{N}_{i,b})_{1\le b\le B}$	Cardinality for $(\widetilde{\mathbb{V}}_{i,b})_{1\le b\le B}$;
$(\mathbb{I}_{i,b})_{1\le b\le B}$	Partition of $\{1,\dots,M_i\}$;
$(M_{i,b})_{1\le b\le B}$	Cardinality for $(\mathbb{I}_{i,b})_{1\le b\le B}$, with $\sum_{b=1}^B M_{i,b} = M_i$.

(A3) The linear operators $\boldsymbol{H}_1\colon \mathbb{R}^N \to \mathbb{R}^{M_1}$ and $\boldsymbol{H}_2\colon \mathbb{R}^N \to \mathbb{R}^{M_2}$ are localized in the sense of Definition 2, with respect to the same partition $(\mathbb{V}_b)_{1\le b\le B}$ of $\{1,\dots,N\}$. Let $(\boldsymbol{S}_{i,b})_{1\le b\le B}$, for $i = 1, 2$, the corresponding families of local selection operators.

(A4) For $i \in \{1,2\}$, the function $f_i\colon \mathbb{R}^{M_i} \to \left(-\infty,+\infty\right]$ is block-additively separable

$$(\forall \boldsymbol{z} \in \mathbb{R}^{M_i}) \quad f_i(\boldsymbol{z}) = \sum_{b=1}^{B} f_{i,b}(\boldsymbol{z}_b), \tag{5}$$

where, for every $b \in \{1,\dots,B\}$, $f_{i,b}\colon \mathbb{R}^{M_{i,b}} \to \left(-\infty,+\infty\right]$, and $\boldsymbol{z}_b = (z^{(m)})_{m\in\mathbb{I}_{i,b}}$, with $(\mathbb{I}_{i,b})_{1\le b\le B}$ the partition of $\{1,\dots,M_i\}$ associated with $\boldsymbol{H}_i$ (see Table I).

(A5) There exists a family $(\boldsymbol{S}_{0,b})_{1\le b\le B}$ of local selection operators with respect to the partition $(\mathbb{V}_b)_{1\le b\le B}$ such that the function $\log p$ in (2) is block-additively separable

$$(\forall \boldsymbol{x} \in \mathbb{R}^N) \quad \log p(\boldsymbol{x}) = \sum_{b=1}^{B} \log p_b(\boldsymbol{S}_{0,b}\boldsymbol{x}). \tag{6}$$

Assumptions (A3)–(A5) are essential to the distributed sampler proposed in Section II, and call for a few remarks.

Remark 1

1) Assumptions (A1) and (A2) are instrumental for Langevin-based transition kernels such as PSGLA [12], MYULA [30] or PnP-ULA [3].
2) Assumption (A3) is motivated by the fact that if $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ are localized operators, there exists a basis where their matrix representation is block-sparse [11]. This is satisfied for many usual inverse problems. Typical examples are convolutions, finite pixel differences appearing in the TV norm [31] or masking operators used in imaging applications (see also the experiments in Section III).
3) When the likelihood is associated with $f_i \circ \boldsymbol{H}_i$, $i = 1$ or 2, Assumption (A4) ensures that the observations $\boldsymbol{y}$ can be partitioned into B conditionally-independent blocks $(y^{(m)})_{m\in\mathbb{I}_{i,b}}$ given $\boldsymbol{x}$. This is in particular the case when f_i comes from the likelihood function associated with an additive white Gaussian noise.

When p in (2) is associated with a denoising network D_ϵ, its structure has an impact not only on estimation performance,

but also on speed and scalability for distributed inference. For an efficient distributed implementation on an SPMD architecture, the structure of D_ϵ should be compatible with Assumption (A5) so that, after lightweight communications, it is fast to evaluate locally on each worker using subsets of locally available data. This is the topic of the next section.

B. Denoiser choice for distributed inference

We focus on CNNs to encode the learned prior p. We describe in this section how their architecture can be distributed using Assumption (A5). In our simulations, we will mainly focus on three state-of-the-art CNNs described in the examples below, namely DnCNN, DDFB and DRUNet.

*1) **Denoising CNNs**:* Let $D_\epsilon \colon \mathbb{R}^N \to \mathbb{R}^N$ be a denoising CNN composed of $K \in \mathbb{N}^*$ layers. In this context, D_ϵ aims to produce a good estimate of an unknown image $\overline{\boldsymbol{x}}$ from a noisy version $\boldsymbol{v} = \overline{\boldsymbol{x}} + \epsilon \boldsymbol{w}$, with $\epsilon > 0$ the standard deviation of the noise, and $\boldsymbol{w}$ a realization of a standard multivariate Gaussian distribution. In other words, $D_\epsilon(\boldsymbol{v}) \approx \overline{\boldsymbol{x}}$. In the following, the input image dimension is $N = C \times N_y \times N_x$, with $C \in \mathbb{N}^*$ the number of image channels and $N_y \times N_x$ its spatial dimensions.

The architecture of D_ϵ can generally be described as a function of $G_\epsilon = T_K \circ \cdots \circ T_1$, where for every $k \in \{1, \dots, K\}$

$$T_k \colon \mathbb{R}^{N_k} \to \mathbb{R}^{N_{k+1}} \colon \boldsymbol{v} \mapsto \eta_k(\boldsymbol{W}_k \boldsymbol{v} + \boldsymbol{b}_k). \tag{7}$$

In (7), the input and output dimensions are $N_k = C_k \times N_{y,k} \times N_{x,k}$ and $N_{k+1} = C_{k+1} \times N_{y,k+1} \times N_{x,k+1}$, respectively, with $N_1 = N_{K+1} = N$. Each layer T_k contains a linear operator $\boldsymbol{W}_k \in \mathbb{R}^{M_k \times N_k}$, with $M_k = P_k \times N_{y,k} \times N_{x,k}$, a bias $\boldsymbol{b}_k \in \mathbb{R}^{M_k}$, and a non-linear activation function $\eta_k \colon \mathbb{R}^{M_k} \to \mathbb{R}^{N_{k+1}}$. For the sake of simplicity, in the following we directly consider the case where the linear operator $\boldsymbol{W}_k$ models a convolution. However, in practice it can also account for skip-connections, by defining $\boldsymbol{W}_k$ as the concatenation of a convolution and the identity operator (such that the current latent variable is carried forward until needed in deeper layers, *e.g.*, for DRUNet described in Example 4). The convolution operator $\boldsymbol{W}_k$ can be expressed, for every $\boldsymbol{v} = (\boldsymbol{v}_c)_{1 \le c \le C_k} \in \mathbb{R}^{N_k}$, as

$$\boldsymbol{W}_k \boldsymbol{v} = \left(\sum_{c=1}^{C_k} \boldsymbol{v}_c * \boldsymbol{w}_{c',c} \right)_{1 \le c' \le P_k} \in \mathbb{R}^{N_k}, \tag{8}$$

where $*$ is a half-padding convolution with kernel $\boldsymbol{w}_{c',c} \in \mathbb{R}^{L_y \times L_x}$ for output feature $c' \in \{1, \dots, P_k\}$ and input feature $c \in \{1, \dots, C_k\}$.

Example 2 (DnCNN denoiser [27]) DnCNN is a simple version of (7), where $N_2 = \cdots = N_K = P \times N_y \times N_x$, and, for every $k \in \{1, \dots, K\}$, $P_k = P$. In other words, the first layer expands the input image from C channels to P features, keeping the same spatial dimension. All the following layers keep the same dimension, except the last layer that reduces from P features to C channels. It can be expressed as

$$D_\epsilon(\boldsymbol{v}) = (\mathrm{Id} - G_\epsilon)(\boldsymbol{v}). \tag{9}$$

In general, convolution kernels are chosen of size $L_y \times L_x = 3^2$, with $K = 17$ or 20 layers. The activation functions are ReLU non-linearities on all layers apart from the last, where no activation function is used (*i.e.*, η_K is identity).

Example 3 (DDFB denoiser) DDFB [24] can be written as a sequential CNN with layers of the form (7), see [25], and is obtained by unrolling a dual forward-backward algorithm [32], [33]. In a nutshell, DDFB is given by

$$D_\epsilon(\boldsymbol{v}) = \mathrm{proj}_{[0,1]^N} \left(\boldsymbol{v} - \gamma_K \boldsymbol{W}_K^* G_{\epsilon,\boldsymbol{v}}(\boldsymbol{W}_K \boldsymbol{v}) \right), \tag{10}$$

with $G_{\epsilon,\boldsymbol{v}} = T_{K-1,\epsilon,\boldsymbol{v}} \circ \cdots \circ T_{1,\epsilon,\boldsymbol{v}}$. Each layer $k \in \{1, \dots, K-1\}$ is the concatenation of two sub-layers: one expanding the dimension of the image from C channels to P features keeping the spatial dimensions fixed (*via* the operator $\boldsymbol{W}_k \in \mathbb{R}^{N \times M}$, with $N = C \times N_y \times N_x$ and $M = P \times N_y \times N_x$), and one reducing back from P features to C channels (*via* the adjoint operator $\boldsymbol{W}_k^* \in \mathbb{R}^{M \times N}$). Specifically, for every $\boldsymbol{u} \in \mathbb{R}^M$,

$$T_{k,\epsilon,\boldsymbol{v}}(\boldsymbol{u}) = \mathcal{HT}_\epsilon \left(\boldsymbol{u} + \gamma_k \boldsymbol{W}_k \, \mathrm{proj}_{[0,1]^N}(\boldsymbol{v} - \boldsymbol{W}_k^* \boldsymbol{u}) \right), \tag{11}$$

where $\mathcal{HT}_\epsilon$ is the element-wise hard-tan operator [25] with hyper-parameter $\epsilon > 0$, $\mathrm{proj}_{\mathcal{C}}$ is a term-wise thresholding operator [33] and $\gamma_k \in (0, 2/\|\boldsymbol{W}_k\|_2^2)$. In general, convolution kernels are chosen of size $L_y \times L_x = 3^2$.

Example 4 (DRUNet denoiser) DRUNet [16] is a CNN with a UNet structure which can be written as (7). It is composed of $K = 2I + 3$ layers, including $I \in \mathbb{N}^*$ downsampling and upsampling layers. The layers $k \in \{2, \dots, K-1\}$ include residual operators $R_k : \mathbb{R}^{N_k} \to \mathbb{R}^{N_k}$, composed of $J \in \mathbb{N}^*$ residual blocks $R_k = R_{k,J} \circ \cdots \circ R_{k,1}$. In the following, the spatial dimensions N_x and N_y are assumed divisible by 2^I (possibly after padding), so that downsampling and upsampling operators can be applied without communications. For $\boldsymbol{v} \in \mathbb{R}^N$, the DRUNet architecture is given by

$$D_\epsilon(\boldsymbol{v}) = \boldsymbol{W}_K \Big(\big(G_\epsilon + \mathrm{Id} \big) \big(\boldsymbol{W}_1 [\boldsymbol{v}, \epsilon \boldsymbol{1}_{N_y \times N_x}] \big) \Big), \tag{12}$$

with $[\boldsymbol{v}, \epsilon \boldsymbol{1}_{N_y \times N_x}] \in \mathbb{R}^{N_\epsilon}$ the concatenation of $\boldsymbol{v}$ and $\epsilon \boldsymbol{1}_{N_y \times N_x}$, $N_\epsilon = (C+1) \times N_y \times N_x$, $\boldsymbol{W}_1 \in \mathbb{R}^{M \times N_\epsilon}$ with $M = P \times N_y \times N_x$, $\boldsymbol{W}_K \in \mathbb{R}^{N \times M}$, and $G_\epsilon = T_{K-1} \circ \cdots \circ T_2 \colon \mathbb{R}^M \to \mathbb{R}^M$. Each layer $k \in \{2, \dots, I+1\}$ is defined as $T_k \colon (\boldsymbol{v}_{k'})_{2 \le k' \le k} \mapsto (\boldsymbol{v}_{k'})_{2 \le k' \le k+1}$. For $k \in \{I+3, \dots, K-1\}$, we have $T_k \colon ((\boldsymbol{v}_{k'})_{2 \le k' \le K-k+1}, \boldsymbol{v}_k) \mapsto ((\boldsymbol{v}_{k'})_{2 \le k' \le K-k}, \boldsymbol{v}_{k+1})$ with $\boldsymbol{v}_{k'} \in \mathbb{R}^{N_{k'}}$ ($N_2 = N_{K-1} = M$), and

$$\boldsymbol{v}_{k+1} = \begin{cases} \boldsymbol{\Phi}_k \circ R_k(\boldsymbol{v}_k), & \text{if } 2 \le k \le I+1, \\ R_{I+2}(\boldsymbol{v}_{I+2}), & \text{if } k = I+2, \\ R_k \circ \boldsymbol{\Psi}_k(\boldsymbol{v}_k + \boldsymbol{v}_{K+2-k}), & \text{if } k \ge I+3, \end{cases} \tag{13}$$

with $\boldsymbol{\Phi}_k$ and $\boldsymbol{\Psi}_k$ downsampling and upsampling operators. The residual blocks are defined, for every $j \in \{1, \dots, J\}$, as

$$R_{k,j} = \big(\boldsymbol{W}_{k,j,2} \circ \eta_k \circ \boldsymbol{W}_{k,j,1} + \mathrm{Id} \big) \colon \mathbb{R}^{N_k} \to \mathbb{R}^{N_k}, \tag{14}$$

where η_k is the ReLU non-linearity, $\boldsymbol{W}_{k,j,1} \colon \mathbb{R}^{N_k} \to \mathbb{R}^{M_k}$ and $\boldsymbol{W}_{k,j,2} \colon \mathbb{R}^{M_k} \to \mathbb{R}^{N_k}$ are convolution operators with $L_y = L_x = 3$ and $P_k = C_k$. Downsampling operators are given by

$$\boldsymbol{\Phi}_k : \boldsymbol{v} \in \mathbb{R}^{N_k} \mapsto \boldsymbol{V}_k \boldsymbol{W}_k \boldsymbol{v} \in \mathbb{R}^{N_{k+1}}, \tag{15}$$

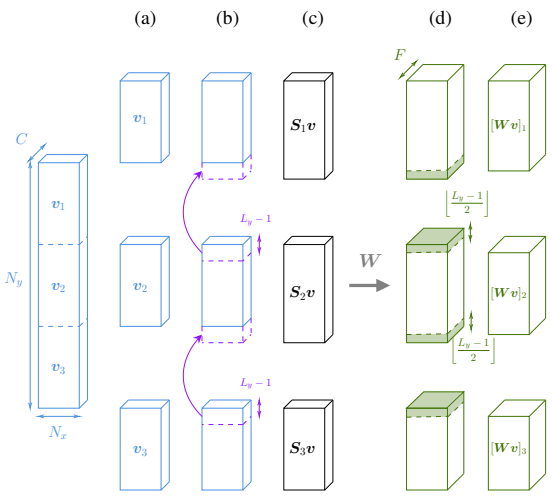

Fig. 2. Communications and operations underlying the proposed distributed implementation of $\boldsymbol{W}_k$. For readability, the layer index k has been omitted from the notation. Communications are represented by purple arrows, with messages in dashed purple lines. Discarded regions are shaded in green.

where $\boldsymbol{W}_k : \mathbb{R}^{N_k} \to \mathbb{R}^{M_k}$ is a convolution operator with $L_y = L_x = 2$, $P_k = 2C_k$, $\boldsymbol{V}_k : \mathbb{R}^{M_k} \to \mathbb{R}^{N_{k+1}}$ is a subsampling operator selecting elements at even indices along both spatial dimensions, and $N_{k+1} = (2C_k) \times \lfloor N_{y,k}/2 \rfloor \times \lfloor N_{x,k}/2 \rfloor$. Upsampling operators are given by

$$\boldsymbol{\Psi}_k : \boldsymbol{u} \in \mathbb{R}^{N_k} \mapsto \boldsymbol{W}_k \boldsymbol{U}_k \boldsymbol{u} \in \mathbb{R}^{N_{k+1}}, \tag{16}$$

where $L_y = L_x = 2$, $P_k = \lfloor C_k/2 \rfloor$, $\boldsymbol{U}_k : \mathbb{R}^{M_k} \to \mathbb{R}^{N_{k+1}}$ inserts a zero between each consecutive entries along the spatial dimensions, and $N_{k+1} = \lfloor C_k/2 \rfloor \times (2N_{y,k}) \times (2N_{x,k})$.

*2) **Proposed distributed implementation**:* We now describe the distributed strategy proposed for (8). It relies on a Cartesian domain decomposition and communications detailed in Section II-B2a. Section II-B2b describes the overlap-save approach [34] used in the proposed distributed implementation.

a) Domain decomposition and communications: For the sake of simplicity, we describe a 1D domain decomposition of $\{1, \dots C\} \times \{1, \dots N_y\} \times \{1, \dots N_x\}$, partitioning only the axis $\{1, \dots N_y\}$. A 2D tessellation can be obtained using the same strategy for the axis $\{1, \dots N_x\}$. To obtain a partition into blocks of similar size, we consider for Assumption (A3) the sets $(\mathbb{V}_b)_{1 \le b \le B}$ with, for every $b \in \{1, \dots, B\}$,

$$\mathbb{V}_b = \{1, \dots, C\} \times \left\{ \left\lfloor \frac{(b-1)N_y}{B} \right\rfloor + 1, \dots, \left\lfloor \frac{bN_y}{B} \right\rfloor \right\} \times \{1, \dots, N_x\}, \tag{17}$$

with $\sharp \mathbb{V}_b = C \times N_{y,b} \times N_x$ and $N_{y,b} = \lfloor bN_y/B \rfloor - \lfloor (b-1)N_y/B \rfloor$. The sets $(\widetilde{\mathbb{V}}_b)_{1 \le b \le B}$ in (8) can be defined as

$$\widetilde{\mathbb{V}}_b = \{1, \dots, C\} \times \left\{ \left\lfloor \frac{(b-1)N_y}{B} \right\rfloor + 1, \dots, \left\lfloor \frac{bN_y}{B} \right\rfloor + L_y - 1 \right\} \times \{1, \dots, N_x\}, \tag{18}$$

with $\sharp \widetilde{\mathbb{V}}_b = \widetilde{N}_b = C \times \widetilde{N}_{y,b} \times N_x$ and $\widetilde{N}_{y,b} = N_{y,b} + L_y - 1$. Note that (17)–(18) satisfy Definition 1 (i) when $\min_{1 \le b \le B} N_b \ge L_y - 1$.

Figure 2 illustrates the partitioning and typical communication patterns in the proposed implementation. In particular, for an image $\boldsymbol{v} \in \mathbb{R}^N$, Figure 2 (a) represents the partition of $\boldsymbol{v}$ into $(\boldsymbol{v}_b)_{1 \le b \le B}$. Figure 2 (b) illustrates the communication pattern considered for the distributed implementation of (8). Each worker $b \in \{2, \dots, B\}$ is required to send a slice covering the first $L_y - 1$ indices along axis N_y (in dashed purple lines) to worker $b-1$, following the communication pattern illustrated by purple arrows. The message received by $b \in \{1, \dots, B-1\}$ is stored in a ghost region [29, Section 7.5.2 p. 306] (in dashed purple lines) to form $\boldsymbol{S}_b \boldsymbol{v} = (v^{(n)})_{n \in \widetilde{\mathbb{V}}_n}$.

b) Distributed implementation for $\boldsymbol{W}_k$: Let $k \in \{1, \dots, K\}$. The operator (8) can be decomposed with an overlap-save approach [34, Section 3.9.2] as follows. For $\boldsymbol{v} \in \mathbb{R}^{N_k}$ and adapting notation in Section II-B1 to the distributed setting from Section II-B2a,

$$\boldsymbol{W}_k \boldsymbol{v} = (\boldsymbol{W}_{k,b} \boldsymbol{S}_{k,b} \boldsymbol{v})_{1 \le b \le B}, \tag{19}$$

where $\boldsymbol{S}_{k,b} \in \{0,1\}^{\widetilde{N}_{k,b} \times N_k}$ with $\widetilde{N}_{k,b} = C_k \times \widetilde{N}_{y,k,b} \times N_{x,k}$, $\widetilde{N}_{y,k,b} = N_{y,k,b} + L_y - 1$, and $\boldsymbol{W}_{k,b} \in \mathbb{R}^{M_{k,b} \times \widetilde{N}_{k,b}}$ with $M_{k,b} = P_k \times N_{y,k,b} \times N_{x,k}$. More precisely, for $\widetilde{\boldsymbol{v}} = (\widetilde{\boldsymbol{v}}_c)_{1 \le c \le C_k} \in \mathbb{R}^{\widetilde{N}_{k,b}}$,

$$\boldsymbol{W}_{k,b} \widetilde{\boldsymbol{v}} = \left(\sum_{c=1}^{C_k} \boldsymbol{A}_{k,b} (\widetilde{\boldsymbol{v}}_c * \boldsymbol{w}_{c',c}) \right)_{1 \le c' \le P_k}, \tag{20}$$

with $\boldsymbol{A}_{k,b} \in \{0,1\}^{(N_{y,k,b} \times N_{x,k}) \times (\widetilde{N}_{y,k,b} \times N_{x,k})}$ a local selection operator. Figure 2 (b)–(d) illustrates the operators in (19)–(20). After communications (see Figure 2 (b)), each worker b can access $\boldsymbol{S}_{k,b} \boldsymbol{v}$ (Figure 2 (c)) for (20). Entries computed with the wrong boundary conditions are discarded via $\boldsymbol{A}_{k,b}$ (Figure 2 (d)), leading to the output blocks in Figure 2 (e).

In the following, we propose to distribute the computations of each successive layer $k \in \{1, \dots, K\}$, communicating the required boundary elements to compute local convolutions (20). In practice, the direction of communications in Figure 2 (b) is reversed from one convolution operator to another to maintain balanced workloads and memory usage for all workers over the sequence of convolutional layers.

Other distributed strategies could be contemplated, such as communicating the receptive field of the full sequence of convolutions computed across the network [35]. This alleviates the need for successive communications phases in each layer, at the cost of a significant increase in the size of messages as the input dimensions and the number of layers increase. Comparing different communication patterns is an interesting perspective beyond the scope of this paper.

C. Proposed distributed sampler

Following the AXDA approach [10], we approximate (2) by introducing an auxilliary variable $\boldsymbol{z} \in \mathbb{R}^{M_2}$ as

$$\pi_\rho(\boldsymbol{x}, \boldsymbol{z} | \boldsymbol{y}) \propto p(\boldsymbol{x}) \exp\Big(- f_1(\boldsymbol{H}_1 \boldsymbol{x}) - f_2(\boldsymbol{z}) - \frac{1}{2\rho} \|\boldsymbol{z} - \boldsymbol{H}_2 \boldsymbol{x}\|_2^2 \Big), \tag{21}$$

where $\rho > 0$ controls the coupling between $\boldsymbol{z}$ and $\boldsymbol{H}_2\boldsymbol{x}$. Note that the same technique could be applied to f_1 as well. Samples can be drawn from (21) with SGS [36], where the associated conditional distributions are

$$\pi_\rho(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{z}) \propto p(\boldsymbol{x}) \exp\big(-f_1(\boldsymbol{H}_1\boldsymbol{x}) - \frac{1}{2\rho}\|\boldsymbol{z}-\boldsymbol{H}_2\boldsymbol{x}\|_2^2\big), \quad (22)$$

$$\pi_\rho(\boldsymbol{z}|\boldsymbol{y},\boldsymbol{x}) \propto \exp\big(-f_2(\boldsymbol{z}) - \frac{1}{2\rho}\|\boldsymbol{z}-\boldsymbol{H}_2\boldsymbol{x}\|_2^2\big). \quad (23)$$

In practice, (22) and (23) can be significantly less expensive to sample from compared to (21). Under Assumptions (A1) and (A2), PSGLA [12] can be used to sample (23). Further, to leverage a pre-trained prior, we propose to use the transition kernel underlying the PnP Unadjusted Langevin Algorithm (PnP-ULA) [3] to update $\boldsymbol{x}^{(t+1)}$ in (22). This approach has the advantage that it ensures the existence of a proper prior distribution with pdf p in (2) associated with a denoiser D_ϵ (pre-trained at noise level $\epsilon > 0$), assuming $\boldsymbol{I}_N - D_\epsilon$ is Lipschitz continuous with constant $L_\epsilon > 0$ [3, Section 3.2].

Hence, for $\kappa \in (0, \rho)$, the transition from iteration $t \in \mathbb{N}^*$ to $t+1$ can be written

$$\begin{aligned}\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} &- \gamma \boldsymbol{H}_1^* \nabla f_1(\boldsymbol{H}_1\boldsymbol{x}^{(t)}) \\ &- \frac{\gamma}{\rho}\boldsymbol{H}_2^*(\boldsymbol{H}_2\boldsymbol{x}^{(t)} - \boldsymbol{z}^{(t)}) + \frac{\alpha\gamma}{\epsilon^2}\big(D_\epsilon(\boldsymbol{x}^{(t)}) - \boldsymbol{x}^{(t)}\big) \\ &+ \frac{\gamma}{\lambda}\big(\operatorname{proj}_{\mathcal{C}}(\boldsymbol{x}^{(t)}) - \boldsymbol{x}^{(t)}\big) + \sqrt{2\gamma}\boldsymbol{\xi}^{(t+1)}, \end{aligned} \quad (24)$$

$$\begin{aligned}\boldsymbol{z}^{(t+1)} = \operatorname{prox}_{\kappa f_2}\big(\boldsymbol{z}^{(t)} &- \frac{\kappa}{\rho}\big(\boldsymbol{z}^{(t)} - \boldsymbol{H}_2\boldsymbol{x}^{(t+1)}\big) \\ &+ \sqrt{2\kappa}\boldsymbol{\zeta}^{(t+1)}\big), \end{aligned} \quad (25)$$

where $\boldsymbol{\xi}^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_N)$, $\boldsymbol{\zeta}^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_{M_2})$, and $\lambda, \alpha, \gamma > 0$ are such that

$$\begin{cases} 2(L + \|\boldsymbol{H}_2\|_2^2) + \alpha\epsilon^{-2}L_\epsilon \leq (2\lambda)^{-1}, \\ 3\gamma(L + \|\boldsymbol{H}_2\|_2^2 + \lambda^{-1} + \alpha\epsilon^{-2}L_\epsilon) < 1. \end{cases} \quad (26)$$

Note that variants of the proposed sampler (24)–(25) can be obtained with other combinations of a PnP-based transition such as [4], with a Langevin kernel relying on a proximal operator, such as MYULA [30], PSGLA [12] or SK-ROCK [37].

The distributed setting of Section II-A paves the way to a distributed counterpart to (24)–(25) inspired by [11], where the denoiser D_ϵ is also distributed, as described in Section II-B. The resulting distributed sampler is reported in Algorithm 1.

Under (A3)–(A4), computations associated with f_1, f_2, $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ can be decomposed over $B \geq 2$ workers. In addition, according to [3, Section 3.2], there exists a proper prior distribution with pdf p in (2) associated with the denoiser D_ϵ. Further, following the proposed distributed strategy for D_ϵ described in Section II-B2, the associated prior p satisfies Assumption (A5). This allows the PnP-ULA (Algorithm 1 lines 5–11) and PSGLA (lines 12–14) steps to be distributed on an SPMD architecture.

Remark 2 A few remarks can be made on the distributed strategy considered in Algorithm 1.

1) Most steps in Algorithm 1 do not require synchronization.
2) Communications are only required between a few workers in lines 5, 8, 9 and 12. The distributed PnP-ULA transition induces communications to compute $\nabla(f_1 \circ \boldsymbol{H}_1)$ (lines 5 and 8) and apply the denoiser (line 9). Some entries in $\boldsymbol{x}^{(t+1)}$ are exchanged between a few workers in line 12 so that $\boldsymbol{z}$ can be updated locally (line 14).
3) The messages in line 5 can be grouped into a single message to reduce communication costs.

Algorithm 1: Proposed distributed sampler.

Input: $\boldsymbol{y} \in \mathbb{R}^M$, $\epsilon, \rho > 0$, and $\kappa \in (0, \rho)$. Choose $\alpha, \lambda, \gamma > 0$ satisfying (26).

1 **for** *each worker* $b \in \{1, \dots, B\}$ **do** in parallel
2 Load and store observation block $\boldsymbol{y}_b \in \mathbb{R}^{M_{1,b}}$;
3 Initialize $\boldsymbol{x}_b^{(0)} \in \mathbb{R}^{N_b}$ and $\boldsymbol{z}_b^{(0)} \in \mathbb{R}^{M_{1,b}}$;
4 **for** $t = 0$ **to** $T-1$ **do**
// Update $\boldsymbol{x}_b$ with PnP-ULA transition [3] (see (24))
5 Communicate to retrieve $\big(\boldsymbol{S}_{i,b}\boldsymbol{x}^{(t)}\big)_{0\leq i\leq 2}$;
6 $\boldsymbol{u}_{1,b}^{(t)} = \boldsymbol{H}_{1,b}^* \nabla f_{1,b}(\boldsymbol{H}_{1,b}\boldsymbol{S}_{1,b}\boldsymbol{x}^{(t)})$;
7 $\boldsymbol{u}_{2,b}^{(t)} = \frac{1}{\rho}\boldsymbol{H}_{2,b}^*(\boldsymbol{H}_{2,b}\boldsymbol{S}_{2,b}\boldsymbol{x}^{(t)} - \boldsymbol{z}_b^{(t)})$;
8 Communicate to compute $\boldsymbol{v}_b^{(t)} = \sum_{j=1}^{B} \boldsymbol{S}_{1,j}^*\boldsymbol{u}_{1,b}^{(t)} + \boldsymbol{S}_{2,j}^*\boldsymbol{u}_{2,j}^{(t)}$;
9 Communicate & compute $\big[D_\epsilon(\boldsymbol{x}^{(t)})\big]_b$ (see Section II-B);
10 Sample $\boldsymbol{\xi}_b^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_{N_b})$;
11 $\boldsymbol{x}_b^{(t+1)} = \boldsymbol{x}_b^{(t)} - \gamma\boldsymbol{v}_b^{(t)} + \frac{\alpha\gamma}{\epsilon^2}\big(\big[D_\epsilon(\boldsymbol{x}^{(t)})\big]_b - \boldsymbol{x}_b^{(t)}\big) + \frac{\gamma}{\lambda}\big(\operatorname{proj}_{\mathcal{C}_b}(\boldsymbol{x}_b^{(t)}) - \boldsymbol{x}_b^{(t)}\big) + \sqrt{2\gamma}\boldsymbol{\xi}_b^{(t+1)}$;
// Update $\boldsymbol{z}_b$ with PSGLA transition [12] (see (25))
12 Communicate to retrieve $\boldsymbol{S}_{2,b}\boldsymbol{x}^{(t+1)}$;
13 Sample $\boldsymbol{\zeta}_b^{(t+1)} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}_{M_{2,b}})$;
14 $\boldsymbol{z}_b^{(t+1)} = \operatorname{prox}_{\kappa f_{2,b}}\big(\boldsymbol{z}_b^{(t)} - \frac{\kappa}{\rho}\big(\boldsymbol{z}_b^{(t)} - \boldsymbol{H}_{2,b}\boldsymbol{S}_{2,b}\boldsymbol{x}^{(t+1)}\big) + \sqrt{2\kappa}\boldsymbol{\zeta}_b^{(t+1)}\big)$;

Output: $(\boldsymbol{x}^{(t)}, \boldsymbol{z}^{(t)})_{1\leq t\leq T}$

III. Image restoration experimental setting

The proposed Algorithm 1 is assessed on three imaging tasks detailed in Section III-A, using three sets of experiments described in Section III-C. Codes to reproduce the experiments will be made available at https://repo/to/code. All the experiments have been conducted on the CNRS supercomputer Jean Zay, hosted by GENCI at IDRIS. Algorithms were run on computing nodes equipped with two 2.5 GHz, 20-core Intel Cascade Lake 6248 processors, and four Nvidia Tesla V100 SXM2 GPUs with 32 GB of memory each.

A. Application setting

Three different applications are considered: image inpainting under additive white Gaussian noise, and image deconvolution under either additive white Gaussian noise or Poisson noise. Synthetic observations have been generated from high-resolution images taken from https://pixabay.com. Ground truth images $\overline{\boldsymbol{x}}$ have been cropped to pre-defined sizes N (see Section III-C), and normalized into $\mathcal{C} = [0,1]^N$. Estimates and pixel-wise variance have been computed from $T = 10^4$ samples from Algorithm 1, including 10^3 burn-in samples.

*1) **Inpainting with Gaussian noise**:* The likelihood is

$$\boldsymbol{y} \mid \boldsymbol{x} \sim \mathcal{N}(\boldsymbol{H}_1\boldsymbol{x}, \sigma^2\boldsymbol{I}_N), \tag{27}$$

with $\boldsymbol{H}_1 = \boldsymbol{H} \in \{0,1\}^{N\times N}$ a diagonal operator encoding the selection mask, and $\sigma^2 > 0$ the noise variance. Observations $\boldsymbol{y}$ are generated with $M = 0.3N$ observed pixels and a noise variance σ^2 leading to a 15 dB input SNR.

The pdf of the posterior distribution is of the form (2), with

$$f_1(\boldsymbol{H}_1\boldsymbol{x}) = \frac{1}{2\sigma^2}\|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}\|_2^2, \tag{28}$$

where the prior is handled through p or $f_2 \circ \boldsymbol{H}_2$. The masking operator acts element-wise, so no communication is needed to evaluate $\boldsymbol{H}_1$ and $\boldsymbol{H}_1^*$ in a distributed computing setting, *i.e.*, $M_{1,b} = N_b$ and $\boldsymbol{S}_{1,b} = \boldsymbol{I}_{N_b}$ in (5).

*2) **Deconvolution with Gaussian noise**:* The likelihood is also of the form (27) with $\boldsymbol{H}_1 = \boldsymbol{H} \in \mathbb{R}^{M\times N}$ a linear convolution operator. A normalized motion blur is considered for $\boldsymbol{H}$, with kernel size $L_x \times L_y$ increasing in the same proportion as N: $(N, L_xL_y) \in \{(3\times 2048^2, 65^2), (3\times 2896^2, 91^2), (3\times 4096^2, 129^2)\}$. Observations are generated with a noise variance σ^2 such that the input SNR is 25 dB.

The pdf of the posterior distribution is of the form (2), (28), with the prior handled through p or $f_2 \circ \boldsymbol{H}_2$. Communications similar to Section II-B are required to compute $\boldsymbol{H}_1$ and $\boldsymbol{H}_1^*$ in a distributed computing setting.

*3) **Deconvolution with Poisson noise**:* The likelihood is

$$\boldsymbol{y} \mid \boldsymbol{x} \sim \mathcal{P}(\eta\boldsymbol{H}\boldsymbol{x}), \tag{29}$$

with $\mathcal{P}$ the Poisson distribution, $\boldsymbol{H}_1 = \boldsymbol{H} \in \mathbb{R}^{M\times N}$ a linear convolution operator, and $\eta > 1$ controlling the variance of the Poisson noise. Observations have been generated with $\eta = 250$ and the same convolution kernel as in Section III-A2.

The pdf of the posterior distribution is of the form (2), where $f_1 \circ \boldsymbol{H}_1 \equiv 0$, and $f_2 \circ \boldsymbol{H}_2$ is split to handle both the pdf of the likelihood and part of the prior (see Section III-B), *i.e.*,

$$f_2(\boldsymbol{H}_2\boldsymbol{x}) = \mathrm{D}_{\mathrm{KL}}(\boldsymbol{y}\|\eta\boldsymbol{H}\boldsymbol{x}) + \widetilde{f}_2(\widetilde{\boldsymbol{H}}_2\boldsymbol{x}), \tag{30}$$

with D_{KL} being the Kullback-Leibler divergence. Both $\widetilde{f}_2 \circ \widetilde{\boldsymbol{H}}_2$ and p will be specified in Section III-B.

As in [11], two blocks are used in the AXDA auxiliary variable $\boldsymbol{z} = (\boldsymbol{z}_1^T, \boldsymbol{z}_2^T)^T$, with coupling parameters $\rho_1, \rho_2 > 0$.

B. Choice of the prior and algorithm setting

Experiments are conducted for the three applications described in section III-A, using two types of priors, both satisfying (A4)–(A5). In our simulations, the sampler is initialized as follows. Auxiliary variables $\boldsymbol{z}$ are initialized to $\boldsymbol{0}$. For the inpainting problem in Section III-A1, $\boldsymbol{x}$ is initialized by interpolating $\boldsymbol{y}$ with bicubic splines. For the deconvolution problems in Sections III-A2 and III-A3, $\boldsymbol{x}$ is initialized to $\boldsymbol{0}$.

*1) **Denoiser-based prior**:* We consider three denoising CNNs, namely DnCNN, DDFB and DRUNet, described in Examples 2, 3 and 4, respectively.

Both DnCNN and DRUNet are pre-trained denoisers, whose weights and implementation are available at https://github.com/cszn/KAIR. The DnCNN network contains $K = 20$ layers, each with $F = 64$ filters. The DRUNet network has been trained with $I = 3$ downsampling/upsampling layers, $J = 4$ blocks per residual layer, and $F = 64$ features. The DDFB network contains $K = 4$ layers, with $F = 64$ latent features. We trained the networks using an ℓ_1-loss, on random patches of size 50×50 from the ImageNet test set [38], corrupted by additive white Gaussian noise with standard deviation ϵ selected uniformly at random between 0 and 0.1. Training has been carried out over 100 epochs using the Adam optimizer [39], with a learning rate of 10^{-3}, a weight decay of 10^{-4} and a batch size of 10^3. The Lipschitz constant L_ϵ of $\boldsymbol{I}_N - D_\epsilon$ has been estimated for each denoiser as in [40] (see Table II for values).

Depending on the noise model (see Section III-A), the functions p and $f_2 \circ \boldsymbol{H}_2$ can be identified as follows.

(i) **Gaussian noise**: We define $f_2 \circ \boldsymbol{H}_2 \equiv 0$, and p as in [3]. The posterior distribution can be directly sampled with PnP-ULA [3] without using AXDA. In the experiments, $\alpha = 1$, $\epsilon = \sigma$, $\lambda = 0.99/(4\|\boldsymbol{H}\|_2^2/\sigma^2 + 2\alpha\epsilon^{-2}L_\epsilon)$ and $\gamma = 0.99/(\alpha L_\epsilon/\epsilon^2 + \|\eta\boldsymbol{H}\|_2^2/\rho + \lambda^{-1})/3$.

(ii) **Poisson noise**: As the likelihood (30) involves a KL divergence, a hybrid regularization is adopted to enforce non-negativity on auxiliary variables after applying AXDA. We define p as in [3], $\widetilde{\boldsymbol{H}}_2 = \boldsymbol{I}_N$ and $\widetilde{f}_2 = \iota_{\mathbb{R}_+^N}$. AXDA is applied as in (21), and the posterior distribution is sampled with Algorithm 1. The PSGLA transition kernel is used to update $\boldsymbol{z} = (\boldsymbol{z}_1^T, \boldsymbol{z}_2^T)^T$ with stepsizes $\kappa_1, \kappa_2 > 0$, respectively. In the experiments, $\alpha = 1$, $\epsilon = 0.05$, $\rho_1 = 10$, $\rho_2 = 10^{-3}$, $\lambda = 0.99/(4\|\eta\boldsymbol{H}\|_2^2/\rho_1 + 4\rho_2^{-1} + 2\alpha\epsilon^{-2}L_\epsilon)$, $\gamma = 0.99/(\alpha L_\epsilon/\epsilon^2 + \|\eta\boldsymbol{H}\|_2^2/\rho_1 + \rho_2^{-1} + \lambda^{-1})/3$, $\kappa_1 = 0.99\rho_1$ and $\kappa_2 = 0.99\rho_2$.

*2) **Isotropic total variation (TV) prior [31]**:* In this case the pdf of the prior is of the form of

$$(\forall \boldsymbol{x} \in \mathbb{R}^N) \quad \pi(\boldsymbol{x}) \propto \exp\big(-\beta\|\boldsymbol{D}\boldsymbol{x}\|_{2,1}\big),$$

with $\beta > 0$ a regularization parameter, $\boldsymbol{D} : \mathbb{R}^N \to \mathbb{R}^{N\times 2}$ the 2D discrete gradient operator, and

$$(\forall \boldsymbol{z} = (\boldsymbol{z}_n)_{1\le n\le N} \in \mathbb{R}^{N\times 2}) \quad \|\boldsymbol{z}\|_{2,1} = \sum_{n=1}^{N} \|\boldsymbol{z}_n\|_2.$$

The functions p and $f_2 \circ \boldsymbol{H}_2$ can be identified as follows.

(i) **Gaussian noise**: In this case, $p = \mathbb{1}_{\mathbb{R}_+^N}$, $f_2 = \beta\|\cdot\|_{2,1}$ and $\boldsymbol{H}_2 = \boldsymbol{D}$. The AXDA approach is applied as in (21). The variable $\boldsymbol{x}$ is updated with the PSGLA [30] transition kernel with stepsize $\gamma > 0$. In the experiments, $\rho = 10^{-5}$, $\beta = 40$, $\gamma = 0.99(\|\boldsymbol{H}_1\|_2^2/\sigma^2 + \|\boldsymbol{D}\|_2^2/\rho)^{-1}$, $\kappa = 0.99\rho\|\boldsymbol{D}\|_2^{-2}$.

(ii) **Poisson noise**: We consider $p = \mathbb{1}_{\mathbb{R}_+^N}$, and $f_2 \circ \boldsymbol{H}_2$ as in (30) with $\widetilde{\boldsymbol{H}}_2 = \boldsymbol{D}$ and $\widetilde{f}_2 = \beta\|\cdot\|_{2,1}$. AXDA is applied as in (21). The variables $\boldsymbol{x}$ and $\boldsymbol{z} = (\boldsymbol{z}_1^T, \boldsymbol{z}_2^T)^T$ are updated with the PSGLA transition kernel (see [11]), with stepsizes γ, κ_1 and $\kappa_2 > 0$, respectively. In the experiments, $\beta = 13$, $\rho_1 = 10$, $\rho_2 = 10^{-3}$, $\gamma = 0.99/(\|\eta\boldsymbol{H}\|_2^2/\rho_1 + \|\boldsymbol{D}\|_2^2/\rho_2)$, $\kappa_1 = 0.99\rho_1$ and $\kappa_2 = 0.99\rho_2$.

TABLE II
DENOISER COMPARISON – NUMBER OF PARAMETERS, MEMORY FOOTPRINT, ESTIMATION OF THE LIPSCHITZ CONSTANT L_ϵ FOR $(\boldsymbol{I}_n - D_\epsilon)$ AND SERIAL RUNTIME.

Denoiser	#Params	Mem. (MiB)	L_ϵ	Time (ms)
DDFB ($K = 4$)	6 912	0.03	2	56.44
DDFB ($K = 20$)	34 560	0.13	2	312.48
DnCNN	668 227	2.55	3	406.27
DRUNet	32 640 960	124.52	7	885.86

TABLE III
GAUSSIAN DENOISING PERFORMANCE – COMPARISON BETWEEN DENOISERS IN TERMS OF OUTPUT QUALITY ($B = 1$ GPU).

Denoiser	SNR	PSNR	SSIM
Noisy input	20.00 ± 0.07	27.80 ± 3.95	0.689 ± 0.196
DDFB ($K = 4$)	27.24 ± 3.09	35.04 ± 4.18	0.928 ± 0.041
DDFB ($K = 20$)	27.61 ± 3.27	35.41 ± 4.24	0.935 ± 0.038
DnCNN	28.54 ± 3.40	36.10 ± 3.90	0.948 ± 0.031
DRUNet	29.13 ± 3.74	36.69 ± 4.19	0.953 ± 0.028

C. Evaluation setting

Performance is evaluated with the following experiments.

1) **Performance and communication cost analysis**: Denoising performance, computation and communication costs of the denoisers are reported in section IV-A. Denoising performance is evaluated on 1000 randomly selected 50×50 patches from DIV2K [41], corrupted with a Gaussian noise so that the input SNR is 20 dB.
2) **Restoration quality experiments**: Estimation quality is discussed in Section IV-B, for $B \in \{1, 2, 4\}$ GPUs and the different choices of priors considered. Reconstruction quality is evaluated with the minimum mean square error (MMSE) estimator. Quality of the estimates is quantified in terms of structural similarity index (SSIM) [42], peak signal-to-noise ratio (PSNR) [43], and SNR defined by

$$\mathrm{SNR}(\overline{\boldsymbol{x}}, \widehat{\boldsymbol{x}}) = 10 \log_{10} \left(\frac{\|\overline{\boldsymbol{x}}\|_2^2}{\|\overline{\boldsymbol{x}} - \widehat{\boldsymbol{x}}\|_2^2} \right), \tag{31}$$

with $\overline{\boldsymbol{x}}$ the ground truth image and $\widehat{\boldsymbol{x}}$ an estimate. Due to memory constraints, uncertainty quantification is reported using the pixel-wise variance of the chain after burn-in[1].
3) **Scalability experiments**: Strong and weak scaling performance of Algorithm 1 are evaluated with one CPU core allocated to each of the B GPUs leveraged in the algorithm. Both are assessed in terms of average runtime per iteration. Strong scaling is assessed with $B \in \{1, 2, 4\}$ GPUs to restore an image of size $N = 3 \times 2048 \times 2048$. Weak scaling is evaluated on problems with image sizes and resources increasing in the same ratio, using $C = 3$ and $(N_y N_x, B) \in \{(2048^2, 1), (2896^2, 2), (4096^2, 4)\}$. Results are reported and discussed in Section IV-C.

TABLE IV
COMMUNICATION EVALUATION – NUMBER OF FLOATING POINT OPERATIONS (FLOPS), MESSAGE SIZES AND COMMUNICATIONS PER WORKER TO DISTRIBUTE THE DIFFERENT PRIORS WHEN PARTITIONING ONLY THE AXIS $\{1, \ldots, N_y\}$ WITH $B \geq 2$ WORKERS.

Prior	FLOPs	Message size	♯ Comms.
TV	$6N_x(N_y/B + 1)$	$6N_x$	1
DDFB ($K = 4$)	$27\,648N_x(N_y/B + 2)$	$67N_x$	8
DnCNN	$1\,336\,451N_x(N_y/B + 2)$	$122N_x$	20
DRUNet	$4\,236\,054N_x(N_y/B + 2)$	$126N_x$	58

IV. EXPERIMENTAL RESULTS

A. Performance and communication cost analysis

Table II compares the denoisers from Section II-B in terms of number of parameters, memory footprint, Lipschitz constant L_ϵ and runtime with $B = 1$ worker. As expected, heavier networks such as DnCNN and DRUNet contain more parameters, have a larger Lipschitz constant and require more runtime. Table III reports the associated performances for Gaussian denoising. It highlights the trade-off between network complexity and performance. In particular, the limited improvements in quality obtained with additional DDFB layers may not justify the corresponding increase in runtime and memory.

Table IV compares the computation and communication costs of the different denoisers using the proposed distributed implementation (see Section II-B2). Computation load is reported as the maximum amount of floating-point operations (FLOPs) carried out by a worker, estimated numerically with the `calflops` library[2]. Message sizes are reported as the average number of elements a worker needs to send in the distributed setting from Section II-B2, which is independent from the number of workers B. Note that splitting along both spatial dimensions N_x and N_y would lead to comparable computing loads and message sizes, with differences mainly in the number of communications (with up to 4 neighbour workers, instead of 2 when partitioning along a single axis).

Although DDFB requires far fewer computations and parameters than the other NNs, its communication costs remain comparable. In fact, only narrow border regions need to be exchanged between workers (1 pixel wide for TV, against $L_y - 1 = 2$ for the CNNs). Dominant factors in communication costs are the number of channels of latent variables for message sizes, and the number of convolutional layers for the number of communication phases.

For DRUNet, although the number of channels doubles after each downsampling layer, both spatial dimensions N_x and N_y are divided by 2: the resulting computation load is thus divided by 2. For N_x, N_y and B fixed, the ratio of FLOPs to message size per worker is similar for DnCNN and DRUNet, and approximately 10 times larger than DDFB. This suggests that DnCNN and DRUNet are more computationally-intensive relative to their communication cost compared to DDFB. As a consequence, larger acceleration factors could be expected for

[1] Pixel-wise variance can be computed while generating the chain, whereas 95% credible intervals require post-processing the samples stored to disk.
[2] Available at https://github.com/MrYxJ/calculate-flops.pytorch

TABLE V
RECONSTRUCTION QUALITY FOR ALL PROBLEMS – BEST AND SECOND BEST ARE HIGHLIGHTED IN BOLD AND UNDERLINED, RESPECTIVELY. THE RESULTS ARE THE SAME FOR ALL $B \in \{1, 2, 4\}$.

Task	Prior	rSNR (↑)	PSNR (↑)	SSIM (↑)
Inpainting	TV	21.12	27.42	0.68
	DDFB ($K = 4$)	<u>22.88</u>	<u>29.19</u>	<u>0.79</u>
	DnCNN	20.76	27.07	0.74
	DRUNet	**23.59**	**29.90**	**0.81**
Gaussian Deconv.	TV	17.56	23.86	0.63
	DDFB ($K = 4$)	**21.05**	**27.35**	**0.79**
	DnCNN	18.93	25.23	0.70
	DRUNet	<u>20.47</u>	<u>26.78</u>	<u>0.74</u>
Poisson Deconv.	TV	<u>19.47</u>	<u>25.78</u>	0.66
	DDFB ($K = 4$)	**20.31**	**26.61**	**0.75**
	DnCNN	18.78	25.08	<u>0.71</u>
	DRUNet	18.52	24.83	0.61

these networks. The scalability of both DnCNN and DRUNet is however mitigated by the larger number of communication phases required (20 and 58) compared to DDFB (8 phases), leading to large runtime overheads.

B. Reconstruction quality experiments

The quality of the MMSE estimates obtained with Algorithm 1 is summarized in Table V. Among the priors tested, DDFB consistently gives estimates with satisfactory and stable metrics, outperforming the TV baseline.

Despite superior performance in pure denoising tasks (see Table III), DnCNN yields poorer estimates. This degradation stems primarily from the denoiser's design: it performs blind denoising, *i.e.*, without explicit regularization parameter ϵ. While DnCNN was trained across a range of noise levels, the PnP-ULA framework assumes a fixed denoising level ϵ throughout the sampling process. The mismatch between the network's implicit behavior and the fixed parameter required by the sampler complicates tuning and may lead to instability, ultimately degrading the reconstruction quality.

DRUNet achieves the best results for the inpainting task, but its performance deteriorates sharply for the deconvolution problem in presence of Poisson noise. These poor metrics come from spurious details and high-frequency artifacts in pixels where uncertainty in the observations is the highest. These artifacts suggest instability during sampling, likely caused by an underestimation of L_ϵ. A more conservative value could stabilize the sampling at the cost of smaller step sizes, reducing the exploration capability of the Markov chain.

For both the inpainting and Gaussian deconvolution tasks, the best results were obtained using the denoising level $\epsilon = \sigma$ for all PnP priors (*i.e.*, the noise level in the observations). In this configuration, the denoiser effectively aligns with the degradation model. For deconvolution under Poisson noise, determining an appropriate denoising level is considerably more challenging. This mostly impacts the reconstruction performance with DRUNet.

Finally, results for the inpainting problem are shown in Figure 3. Restored images for the deconvolution problems can be found in the supplementary material, as well as reconstruction

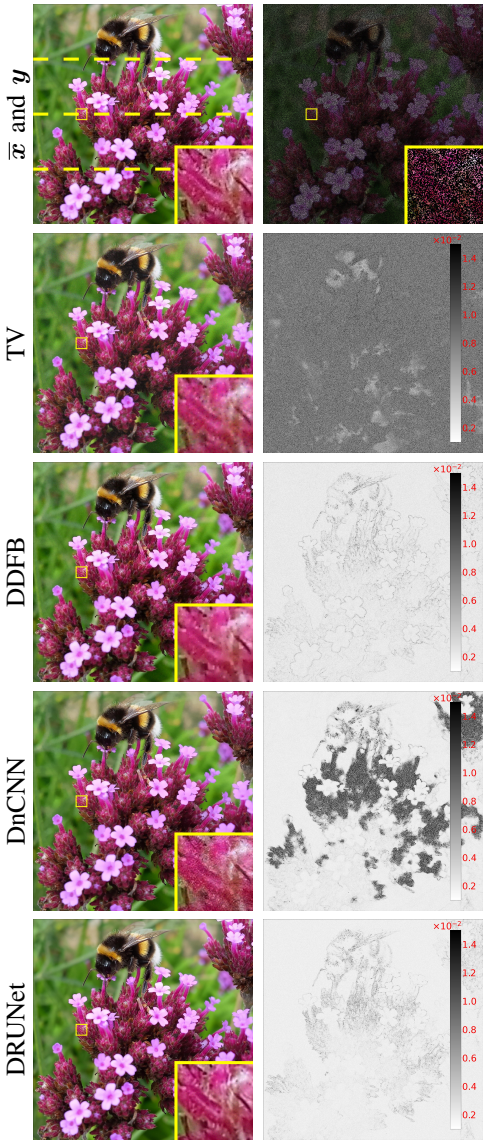

Fig. 3. **Inpainting results –** Restoration results with Algorithm 1 using $B = 4$ workers. First row: ground truth $\overline{\boldsymbol{x}}$ and observations $\boldsymbol{y}$. Last 4 rows, top to bottom: MMSE (left) and pixel variance of the red channel (right) using either TV, DDFB, DnCNN, or DRUNet. Dashed lines on the ground truth image show the partition considered for $\boldsymbol{x}$.

error maps for the estimates. Pixels with the highest variance also have the highest errors. The MMSE and pixel-wise variance estimates are displayed for the inpainting task of an $N = 3 \times 2048^2$ image using $B = 4$ worker (see Section III-A1). Note that Algorithm 1 produces estimates of the same quality for $B \in \{1, 2, 4\}$. In particular, deterministic+ operation outputs coincide for any number of workers B down to machine precision, except the generation of pseudo-random numbers.

C. Scalability experiments

Both strong and weak scaling results are influenced by the overall balance between computing and communication costs. In particular, operators in the likelihood term can significantly affect scalability. For inpainting, all operations in the likelihood are element-wise and fully local (*i.e.*, no communication

TABLE VI
STRONG SCALING EXPERIMENTS – RUNTIME PER ITERATION (IN MS) AND SPEED-UP FOR $B \in \{2, 4\}$ WITH RESPECT TO $B = 1$. RESULTS IN RED HIGHLIGHT CONFIGURATIONS FOR WHICH THERE IS NO PRACTICAL BENEFIT FROM A DISTRIBUTED OVER A SERIAL IMPLEMENTATION.

Task	**Prior**	**Time in ms (speed-up w.r.t $B = 1$)**		
		$B = 1$	$B = 2$	$B = 4$
Inpainting	TV	7.09	7.18 (0.99)	4.17 (1.40)
	DDFB	88.97	65.09 (1.37)	36.80 (2.42)
	DnCNN	342.13	241.40 (1.42)	176.29 (1.94)
	DRUNet	1505.90	998.22 (1.51)	771.46 (1.95)
Gaussian Deconv.	TV	11.71	21.16 (0.55)	24.32 (0.48)
	DDFB	94.84	80.75 (1.17)	57.08 (1.66)
	DnCNN	353.00	251.72 (1.40)	196.05 (1.80)
	DRUNet	1499.61	1001.38 (1.50)	791.41 (1.89)
Poisson Deconv.	TV	14.04	24.24 (0.58)	25.11 (0.56)
	DDFB	97.56	84.05 (1.16)	59.82 (1.63)
	DnCNN	356.75	255.86 (1.39)	197.11 (1.81)
	DRUNet	1514.14	1122.49 (1.35)	793.18 (1.91)

TABLE VII
WEAK SCALING EXPERIMENTS – RUNTIME PER ITERATION (IN MS) AND EFFICIENCY FOR $(B, N_y N_x) \in \{(1, 2048^2), (2, 2896^2), (4, 4096^2)\}$. EFFICIENCY RESULTS INDICATING POOR USE OF HARDWARE RESOURCES (BELOW 30%) ARE HIGHLIGHTED IN RED.

Task	**Prior**	**Time in ms (efficiency w.r.t. setting $B = 1$)**		
		$(1, 2048^2)$	$(2, 2896^2)$	$(4, 4096^2)$
Inpainting	TV	7.09	13.87 (51%)	14.47 (49%)
	DDFB	88.97	127.30 (70%)	186.37 (47%)
	DnCNN	342.13	468.51 (73%)	540.66 (63%)
	DRUNet	1505.90	1891.94 (80%)	2180.04 (69%)
Gaussian Deconv.	TV	11.71	43.86 (27%)	92.73 (13%)
	DDFB	94.84	159.76 (59%)	268.02 (35%)
	DnCNN	353.00	500.53 (71%)	620.98 (57%)
	DRUNet	1499.61	1931.7 (78%)	2249.29 (67%)
Poisson Deconv.	TV	14.04	48.12 (29%)	98.24 (14%)
	DDFB	97.56	173.99 (56%)	277.35 (35%)
	DnCNN	356.75	509.76 (70%)	628.69 (57%)
	DRUNet	1514.14	1958.7 (77%)	2260.03 (67%)

is required). In contrast, the likelihood for deconvolution tasks induce communication costs scaling with the kernel size. This distinction has an impact on scalability, especially for tasks based on a prior with a low computation cost, such as the TV.

1) Strong scaling: Table VI reports strong scaling results for the applications described in section III-A. Speed-up values below 1, highlighted in red, indicate that distributing over multiple workers does not lead to any acceleration (hence do not justify the use of multiple GPUs). This is the case when using the TV prior: apart from the inpainting experiment with $B = 4$, none of the other experiments benefit from the distributed strategy. Instead, samplers based on deep priors exhibit good scaling behaviour due to a favorable ratio between computation and communication costs, with speed-ups between 1.1 and 1.5 for $B = 2$, and 1.63 and 2.42 for $B = 4$.

2) Weak scaling: Table VII presents weak scaling results, obtained by increasing both B and $N_y N_x$ in the same proportion. Ideal weak scaling corresponds to a constant runtime over the different configurations. However, in practice, a gradual degradation in efficiency is often expected as B increases, due to increasing communication overheads.

As for strong scaling, samplers based on deep priors yield better efficiency than TV. With the latter, local computation costs do not significantly outweight communication overheads. For inpainting, all denoisers maintain acceptable efficiency levels (between 70% and 80%), while TV drops to 51% for $B = 2$. For deconvolution problems, communication costs related to the likelihood increase significantly, as the kernel size increases with the image size. This results in sharp efficiency drops across all priors: below 60% for DnCNN and DDFB for $B = 2$, and below 40% for $B = 4$. The TV prior suffers the most, with 13% efficiency with $B = 4$.

Overall, scalability results confirm the interest of the proposed distributed approach when the computing load on each worker is large enough to justify the communication costs incurred, as for other distributed computing approaches [29].

V. CONCLUSION AND PERSPECTIVES

To address high-dimensional inverse problems exceeding the memory of a single computing device, this work introduces a distributed PnP Split Gibbs sampler implemented on a Single Program Multiple Data architecture. It builds on PnP-ULA [3] to leverage a prior based on a lightweight CNN denoiser. The proposed sampler exploits the locality of the operators involved in the algorithm for an efficient implementation on a multi-GPU architecture, numerically equivalent to its serial counterpart.

Experiments on high-dimensional inpainting and deconvolution problems illustrated the quality of the estimates obtained with the proposed approach, comparing different choices of denoisers. The proposed distributed implementation exhibits both strong and weak scaling behaviour. Moreover, it enabled efficient sampling of high-resolution 4096^2 color images, impossible to address with a serial sampler.

Future work will compare different communication strategies [35], and extend the approach to imaging problems based on non-local operators, affected by severe load-imbalance in a distributed setting. High-dimensional problems with data defined on graphs or hypergraphs will also be investigated.

REFERENCES

[1] C. P. Robert, *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation*, 2nd ed., ser. Springer Texts in Statistics. Springer, 2007.

[2] M. Holden, M. Pereyra, and K. C. Zygalakis, "Bayesian imaging with data-driven priors encoded by neural networks," *SIAM J. Imaging Sci.*, vol. 15, no. 2, pp. 892–924, Jun. 2022.

[3] R. Laumont, V. de Bortoli, A. Almansa, J. Delon, A. Durmus, and M. Pereyra, "Bayesian imaging using Plug & Play priors: When Langevin meets Tweedie," *SIAM J. Imaging Sci.*, vol. 15, no. 2, pp. 701–737, Jun. 2022.

[4] E. C. Faye, M. D. Fall, and N. Dobigeon, "Regularization by denoising: Bayesian model and Langevin-within-split Gibbs sampling," *IEEE Trans. Image Process.*, vol. 34, pp. 221–234, 2025.

[5] T. I. Liaudat, M. Mars, M. A. Price, M. Pereyra, M. M. Betcke, and J. D. McEwen, "Scalable Bayesian uncertainty quantification with data-driven priors for radio interferometric imaging," *RAS Tech. and Instrum.*, vol. 3, no. 1, pp. 505–534, Jan. 2024.

[6] N. Komodakis and J.-C. Pesquet, "Playing with duality: an overview of recent primal-dual approaches for solving large-scale optimization problems," *IEEE Signal Process. Mag.*, vol. 32, no. 3, pp. 31–54, Nov. 2015.

[7] T. T.-K. Lau, H. Liu, and T. Pock, "Non-log-concave and nonsmooth sampling via Langevin Monte Carlo algorithms," in *Proc. Adv. Tech. in Optim. for Mach. Learning and Imaging*, A. Benfenati, F. Porta, T. A. Bubba, and M. Viola, Eds. Singapore: Springer Nature, 2024, pp. 83–149.

[8] M. Burger, M. J. Ehrhardt, L. Kuger, and L. Weigand, "Analysis of primal-dual Langevin algorithms," Nov. 2024, arXiv preprint, 10.48550/arXiv.2405.18098.

[9] D. Geman and C. Yang, "Nonlinear image recovery with half-quadratic regularization," *IEEE Trans. Image Process.*, vol. 4, no. 7, pp. 932–946, 1995.

[10] M. Vono, N. Dobigeon, and P. Chainais, "Asymptotically exact data augmentation: Models, properties, and algorithms," *J. Comput. Graph. Stat.*, vol. 30, no. 2, pp. 335–348, 2021.

[11] P.-A. Thouvenin, A. Repetti, and P. Chainais, "A distributed split-Gibbs sampler with hypergraph structure for high-dimensional inverse problems," *J. Comput. Graph. Stat.*, vol. 33, no. 3, pp. 814–832, Oct. 2024.

[12] A. Salim, D. Koralev, and P. Richtarik, "Stochastic proximal Langevin algorithm: Potential splitting and nonasymptotic rates," in *Adv. in Neural Information Process. Systems*, vol. 32. Curran Associates, Inc., 2019, pp. 6649–6661.

[13] F. Darema, "The SPMD model: Past, present and future," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Y. Cotronis and J. Dongarra, Eds., Heidelberg, 2001, p. 1.

[14] G. Ongie, A. Jalal, C. A. Metzler, R. G. Baraniuk, A. G. Dimakis, and R. Willett, "Deep learning techniques for inverse problems in imaging," *IEEE J. Sel. Inf. Theory*, vol. 1, no. 1, pp. 39–56, May 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9084378/?arnumber=9084378

[15] S. V. Venkatakrishnan, C. A. Bouman, and B. Wohlberg, "Plug-and-Play priors for model based reconstruction," in *Proc. IEEE Glob. Conf. on Sig. and Inf. Processing (GlobalSIP)*, Dec. 2013, pp. 945–948. [Online]. Available: https://ieeexplore.ieee.org/document/6737048/?arnumber=6737048

[16] K. Zhang, Y. Li, W. Zuo, L. Zhang, L. Van Gool, and R. Timofte, "Plug-and-Play image restoration with deep denoiser prior," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 10, pp. 6360–6376, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9454311

[17] Z. Cai, J. Tang, S. Mukherjee, J. Li, C. Schönlieb, and X. Zhang, "NF-ULA: Normalizing flow-based unadjusted Langevin algorithm for imaging inverse problems," *SIAM J. Imaging Sci.*, vol. 17, no. 2, pp. 820–860, Jun. 2024.

[18] F. Coeurdoux, N. Dobigeon, and P. Chainais, "Plug-and-Play split Gibbs sampler: Embedding deep generative priors in Bayesian inference," *IEEE Trans. Image Process.*, vol. 33, pp. 3496–3507, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10541919?source=authoralert

[19] Z. Wu, Y. Sun, Y. Chen, B. Zhang, Y. Yue, and K. L. Bouman, "Principled probabilistic imaging using diffusion models as Plug-and-Play priors," *Adv. in Neural Information Process. Systems*, vol. 37, pp. 118 389–118 427, Dec. 2024. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/hash/d65c4ce22241138c1784ff753d4c746c-Abstract-Conference.html

[20] F. Fleuret, "The little book of deep learning," 2024. [Online]. Available: https://fleuret.org/public/lbdl.pdf

[21] I. Kobyzev, S. J. D. Prince, and M. A. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 11, pp. 3964–3979, Nov. 2021.

[22] G. Daras, H. Chung, C.-H. Lai, Y. Mitsufuji, J. C. Ye, P. Milanfar, A. G. Dimakis, and M. Delbracio, "A survey on diffusion models for inverse problems," Sep. 2024, arXiv preprint, 10.48550/arXiv.2410.00083.

[23] V. Monga, Y. Li, and Y. C. Eldar, "Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing," *IEEE Signal Process. Mag.*, vol. 38, no. 2, pp. 18–44, Mar. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9363511

[24] A. Repetti, M. Terris, Y. Wiaux, and J.-C. Pesquet, "Dual forward-backward unfolded network for flexible Plug-and-Play," in *Proc. European Signal Process. Conf. (EUSIPCO)*. Belgrade, Serbia: IEEE, Aug. 2022, pp. 957–961.

[25] H. T. V. Le, A. Repetti, and N. Pustelnik, "Unfolded proximal neural networks for robust image Gaussian denoising," *IEEE Trans. Image Process.*, vol. 33, pp. 4475–4487, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10630640/?arnumber=10630640

[26] M. Bouton, P.-A. Thouvenin, A. Repetti, and P. Chainais, "Multi-GPU distributed PnP-ULA for high-dimensional imaging inverse problems," in *Proc. IEEE Workshop Stat. Sign. Proc.*, Edinburgh, U. K., Jun. 2025, pp. 66–70.

[27] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising," *IEEE Trans. Image Process.*, vol. 26, no. 7, pp. 3142–3155, Jul. 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7839189

[28] J. J. Moreau, "Proximité et dualité dans un espace hilbertien," *Bulletin de la société mathématique de France*, vol. 93, p. 273, 1965. [Online]. Available: https://hal.science/hal-01740635

[29] V. Eijkhout, "The science of computing," 2023. [Online]. Available: https://github.com/VictorEijkhout/TheArtofHPC_pdfs

[30] A. Durmus, E. Moulines, and M. Pereyra, "Efficient Bayesian computation by proximal Markov chain Monte Carlo: When Langevin meets Moreau," *SIAM J. Imaging Sci.*, vol. 11, no. 1, pp. 473–506, Jan. 2018.

[31] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Phys. D*, vol. 60, no. 1-4, pp. 259–268, Nov. 1992.

[32] P. L. Combettes, D. Dũng, and B. C. Vũ, "Dualization of signal recovery problems," *Set-Valued Anal*, vol. 18, no. 3, pp. 373–404, Dec. 2010.

[33] P. L. Combettes and J.-C. Pesquet, "Proximal splitting methods in signal processing," in *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*. Springer, May 2011, vol. 49, pp. 185–212.

[34] M. Vetterli, J. Kovačević, and V. K. Goyal, *Foundations of Signal Processing*. Cambridge: Cambridge University Press, 2014.

[35] B. Galerne, L. Raad, J. Lezama, and J.-M. Morel, "Scaling painting style transfer," *Computer Graphics Forum*, vol. 43, no. 4, p. e15155, 2024.

[36] M. Vono, N. Dobigeon, and P. Chainais, "Split-and-augmented Gibbs sampler–Application to large-scale inference problems," *IEEE Trans. Signal Process.*, vol. 67, no. 6, pp. 1648–1661, Mar. 2019.

[37] M. Pereyra, L. V. Mieles, and K. C. Zygalakis, "Accelerating proximal Markov chain Monte Carlo by using an explicit stabilized method," *SIAM J. Imaging Sci.*, vol. 13, no. 2, pp. 905–935, Jan. 2020.

[38] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[39] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Jan. 2017, arXiv preprint, 10.48550/arXiv.1412.6980.

[40] J.-C. Pesquet, A. Repetti, M. Terris, and Y. Wiaux, "Learning maximally monotone operators for image recovery," *SIAM J. Imaging Sci.*, vol. 14, no. 3, pp. 1206–1237, 2021.

[41] E. Agustsson and R. Timofte, "NTIRE 2017 challenge on single image super-resolution: Dataset and study," in *Proc. IEEE Conf. Comput. Vis. and Pattern Recog. Workshops (CVPRW)*, Jul. 2017, pp. 1122–1131.

[42] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004. [Online]. Available: http://ieeexplore.ieee.org/document/1284395/

[43] Z. Wang and A. C. Bovik, "Mean squared error: Love it or leave it? A new look at signal fidelity measures," *IEEE Signal Process. Mag.*, vol. 26, no. 1, pp. 98–117, Jan. 2009.