A distributed Gibbs Sampler with Hypergraph Structure for High-Dimensional Inverse Problems

P.-A. Thouvenin‡, A. Repetti†, P. Chainais‡ *

‡ Université de Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
† School of Mathematics and Computer Sciences, Heriot-Watt University, Edinburgh, UK
⋆ School of Engineering and Physical Sciences, Heriot-Watt University, Edinburgh, UK

{pierre-antoine.thouvenin, pierre.chainais}@centralelille.fr, a.repetti@hw.ac.uk

Abstract

Sampling-based algorithms are classical approaches to perform Bayesian inference in inverse problems. They provide estimators with the associated credibility intervals to quantify the uncertainty on the estimators. Although these methods hardly scale to high dimensional problems, they have recently been paired with optimization techniques, such as proximal and splitting approaches, to address this issue. Such approaches pave the way to distributed samplers, splitting computations to make inference more scalable and faster. We introduce a distributed Gibbs sampler to efficiently solve such problems, considering posterior distributions with multiple smooth and non-smooth functions composed with linear operators. The proposed approach leverages a recent approximate augmentation technique reminiscent of primal-dual optimization methods. It is further combined with a block-coordinate approach to split the primal and dual variables into blocks, leading to a distributed block-coordinate Gibbs sampler. The resulting algorithm exploits the hypergraph structure of the involved linear operators to efficiently distribute the variables over multiple workers under controlled communication costs. It accommodates several distributed architectures, such as the Single Program Multiple Data and client-server architectures. Experiments on a large image deblurring problem show the performance of the proposed approach to produce high quality estimates with credibility intervals in a small amount of time.

Keywords: MCMC algorithm, Bayesian inference, block-coordinate algorithm, distributed architecture, high dimensional

*PC and PAT acknowledge support from the ANR project “Chaire IA Sherlock” ANR-20-CHIA-0031-01, the programme d’investissements d’avenir ANR-16-IDEX-0004 ULNE and Région HDF. AR acknowledges support from the Royal Society of Edinburgh.
1 Introduction

An inverse problem consists in inferring an unknown variable $x \in H$ from degraded observations $y \in H$. In the remainder, we will consider that $H = \mathbb{R}^N$ and $G = \mathbb{R}^M$. Observations and parameters are typically related by a model of the form

$$y = D(Ax),$$

where the linear operator $A$ models the acquisition process, and $D$ models random perturbations damaging the observations. Bayesian inference relies on the posterior distribution of the random variable $x$ to estimate the true value $\pi$. The posterior distribution combines information from the likelihood – related to the observations $y$ – and the prior. For instance, in image processing, a usual choice consists in promoting sparsity in a selected basis, e.g., a gradient basis leading to the total variation (TV) regularization [28], or a wavelet basis [22]. Prior information can also encompass constraints such as nonnegativity, e.g., for intensity images [5, 30], or more evolved constraints based on the physics of the data acquisition process, e.g., polarization constraints [3]. This work focuses on inverse problems whose statistical description is encoded by a generic posterior distribution of the form

$$\pi(x) \propto \exp(-h(x) - f(x) - g(Dx)),$$

where $h: H \rightarrow ]-\infty, +\infty]$ is a Lipschitz-differentiable function, $f: H \rightarrow ]-\infty, +\infty]$, and $D: H \rightarrow G$ is a linear operator. A classical inference approach is to form the maximum a posteriori (MAP) estimator, defined as a solution to

$$\min_{x \in H} h(x) + f(x) + g(Dx).$$

The problem can be efficiently solved with proximal primal-dual methods [19, 13, 36]. These approaches recast the problem (1.3) as a saddle point problem in order to simplify computations associated with the composite function $g \circ D$. Splitting proximal algorithms benefit from many acceleration techniques (e.g., inertia [2, 23], preconditioning [7, 12]), they are versatile, and scalable. In particular, they are highly parallelizable, and can be efficiently distributed to split the computational cost per iteration, under well-established theoretical guarantees [1, 10, 19]. However, these algorithms only provide a point estimate, without additional information. In absence of ground truth, these approaches do not allow the uncertainty about the estimate to be directly quantified.

Markov chain Monte Carlo (MCMC) algorithms are generic approaches providing estimates with associated credibility intervals [27]. They aim to generate a Markov chain that yields samples from the target posterior distribution (1.2) in the stationary regime. Nevertheless, they are often considered computationally too expensive to handle high dimensional problems, especially when composite functions are involved. Over the last decade, multiple authors have proposed more versatile and scalable optimization-inspired MCMC algorithms [16, 24, 29]. These approaches exploit quantities repeatedly used in optimization, such as gradients and proximity operators, to
efficiently explore high dimensional parameter spaces. In particular, a splitting approach based on an asymptotically exact data augmentation (AXDA) scheme has recently been proposed in [34, 35]. Reminiscent of primal-dual optimization approaches, AXDA introduces auxiliary variables to split composite distributions. The distribution with density (1.2) is approximated by

\[ \pi_{(\alpha,\beta)}(x, z, u) \propto \exp \left( -h(x) - f(x) - g(z) - \phi_\alpha(Dx, z - u) - \psi_\beta(u) \right), \]  

where \( \phi_\alpha : \mathbb{G} \times \mathbb{G} \to [-\infty, +\infty], \) \( \psi_\beta : \mathbb{G} \to [-\infty, +\infty], \) and \( (\alpha, \beta) \) controls the discrepancy between \( Dx \) and \( z. \) For appropriate choices of \( \phi_\alpha \) and \( \psi_\beta, \) the marginal distribution of \( x \) with respect to (1.4) converges to the target distribution (1.2) as \( (\alpha, \beta) \to (0, 0). \) A block Gibbs sampler is proposed in [34, 35] to draw samples from (1.4), referred to as the Split-and-Augment (SPA) Gibbs sampler.

Designing efficient algorithms to handle distributions of the form (1.2) becomes more challenging when multiple composite functions are considered, i.e.,

\[ (\forall x \in \mathbb{H}) \quad g(Dx) = \sum_{i=1}^{I} g_i(D_i x), \]  

and when the dimension of \( x \) and \( y \) increases. On the one hand, primal-dual proximal algorithms [13, 36, 19, 25, 9] enable composite terms (1.5) to be handled in parallel in the dual domain. In this context, the dual variables can be distributed on multiple workers. This strategy has also been adopted in [34] to design an SPA Gibbs sampler tailored to distributions involving composite terms (1.5). On the other hand, to address problems with high dimensional variable \( x, \) a usual strategy in optimization consists in splitting \( x \) into blocks \( (x_k)_{1 \leq k \leq K} \) and either alternate between the blocks [4, 8, 21, 32], or distribute the blocks over multiple workers [11, 25]. In particular, in [25], the authors combine such block-coordinate approaches with primal-dual splitting techniques to parallelize and distribute both the primal and dual variables. Then, the associated minimisation problem is of the form

\[ \minimize_{x=(x_k)_{1 \leq k \leq K} \in \mathbb{H}} \sum_{k=1}^{K} h_k(x_k) + f_k(x_k) + \sum_{i=1}^{I} \sum_{k=1}^{K} D_{i,k} x_k, \]  

where, for every \( k \in \{1, \ldots, K\}, \) \( h_k \) and \( f_k \) only act on the \( k \)-th block of the variable \( x \in \mathbb{H}. \) In [25], the authors also combine the resulting algorithms with consensus constraints, imposing that all \( (x_k)_{1 \leq k \leq K} \) are equal, to develop asynchronous distributed algorithms over hypergraph structures, allowing high flexibility in the choice of the distribution architecture.

Only a few distributed samplers have been proposed in the literature [26, 34, 35]. However, these distributed samplers only focus on distributing the dual variables associated with the composite functions (1.5), without splitting the global variable into blocks. In particular, in [26], a consensus-based approximate posterior distribution has been proposed, addressed with a distributed Metropolis-within-Gibbs sampler relying on a client-server architecture. The sampler exploits the conditional independence between some variables to parallelize computations. This is
especially relevant for data-distributed applications, in which a shared parameter value needs to be inferred from a dataset distributed over multiple workers. However, this setting has several drawbacks. First, the consensus constraint necessitates the variables of interest to be duplicated on the different workers, which can be a critical issue in a high dimensional setting. Second, the client-server architecture may induce communication bottlenecks, as all (clients) workers need to communicate with the server. It also exhibits limitations in terms of distribution flexibility, since it only applies to cases with more than two composite functions, without splitting the global variable of interest $x$ into blocks.

In this work, we introduce a distributed block-coordinate SPA-Gibbs sampler, splitting the global variable of interest into blocks $(x_k)_{1 \leq k \leq K}$, and able to handle multiple composite terms of the form (1.5). Each block $x_k$ is associated with a worker, and communications between the workers are characterized by the structure of the linear operators $(D_i)_{1 \leq i \leq I}$. Precisely, the communications are characterized by the hyperedges of hypergraphs whose adjacency matrices are prescribed by each of the linear operators. For the sake of clarity and simplicity of the presentation, the main part of the paper presents the proposed distributed algorithm for the probability distribution (1.2). The proposed approach is then extended in the appendix to distributions involving multiple linear operators, as in (1.5). Note that the hypergraph structure of the proposed algorithm is different from the one used in [25]. Indeed, we propose to use the hypergraph to distribute the communications induced by $D$ with respect to both the primal and dual variable, while in [25], the hypergraph structure is used when multiple $(D_i)_{1 \leq i \leq I}$ are considered without splitting the primal variable into blocks $(x_k)_{1 \leq k \leq K}$.

The resulting proposed algorithm accommodates several distributed architectures, and is particularly suitable for a Single Program Multiple Data (SPMD) architecture [15]. In contrast with a client-server configuration, all the workers involved in an SPMD architecture execute similar tasks on a subpart of all the variables. An SPMD architecture can drastically reduce the communication.

Figure 1: Example of the block-sparse structure of the matrix $D$ involved in (1.2). The columns of $D$ are split into $K$ contiguous blocks (black dashed lines), with a small overlap compared to the size of the blocks. The orange dashed rectangles highlight the subparts of $D$ implemented on each worker $k$. These act on a parameter block stored on worker $k$, and a few parameters stored on worker $k+1$. 

costs compared to a client-server architecture when a small number of workers are involved in
the communications, and especially when the hypergraph structure of \( D \) induces localized
couplings between parameters. A particular case consists in considering a block-sparse matrix \( D \), as
illustrated in Figure 1. This case is encountered in applications such as image deconvolution or
inpainting, or when considering local priors such as the TV norm. When multiple composite terms
are considered as in (1.5), the proposed algorithm is also amenable to a parallel implementa-
tion based on a client-server architecture, exploiting conditional independence between blocks of
variables. As previously emphasized, this configuration may however suffer from communication
bottlenecks, with a number of workers limited to the number of conditionally independent blocks
of variables in the model. In contrast, the SPMD approach allows a larger number of workers to
be used by exploiting the structure of the hypergraphs.

The remainder of the paper is organized as follows. The AXDA approach and SPA-Gibbs sam-
pler [34, 35] are recalled in Section 2. The proposed hypergraph model is introduced in Section 4,
and the associated distributed block-coordinate SPA-Gibbs sampler is given in Section 5. Section 6
describes an SPMD implementation of the proposed method for a large scale image deconvolu-
tion problem. Conclusion and perspectives are given in Section 7. Eventually, in Appendix A the
proposed method is extended to distributions involving multiple composite terms.

2 AXDA and SPA Gibbs samplers

In this section, we summarize the approximation results at the basis of the AXDA approach [34].
We also describe the SPA Gibbs sampler to approximately draw samples from the distribution (1.2).

2.1 AXDA splitting approach

Consider the distribution \( \pi_{(\alpha,\beta)}(x, z, u) \) introduced in (1.4). Under technical conditions on the
functions \( \phi_\alpha \) and \( \psi_\beta \), [34] showed that the marginal distribution of \( x \) can be made arbitrarily close
to the original distribution \( \pi(x) \) given in (1.2), depending on the choice of \( (\alpha, \beta) \). This result is
summarized in the following proposition.

**Proposition 2.1** [34, Thm. 1 and 2] Let \( (\alpha, \beta) \in [0, +\infty]^2, \phi_\alpha : G \times G \rightarrow ]-\infty, +\infty[ \) and \( \psi_\beta : G \rightarrow ]-\infty, +\infty[ \).
Assume that

\[
(\forall x \in H) \quad \lim_{\alpha \to 0} \frac{\exp \left( -\phi_\alpha(Dx, z) \right)}{\int_G \exp \left( -\phi_\alpha(Dx, z) \right) dz} = \delta_x(z), \tag{2.1}
\]

and that, there exists \( \eta_{\alpha,\beta} \in ]0, +\infty[ \),

\[
(\forall (x, z) \in H \times G) \quad \int_G \exp \left( -\phi_\alpha(Dx, z - u) - \psi_\beta(u) \right) du \propto \exp \left( -\phi_{\eta_{\alpha,\beta}}(Dx, z) \right). \tag{2.2}
\]

Let, for \( x \in H, \pi_{\eta_{\alpha,\beta}}(x) = \int_{G \times G} \pi_{\alpha,\beta}(x, z, u)dudz. \) Then, \( \| \pi - \pi_{\eta_{\alpha,\beta}} \|_{TV} \to 0 \) when \( \eta_{\alpha,\beta} \to 0. \)
Theorem 2.1 ensures that the marginal distribution of $x$ with respect to $\pi(\alpha, \beta)$ converges to the target distribution $\pi$ given in (1.2), provided conditions (2.1) and (2.2) hold. In particular, in [34], these conditions are shown to be satisfied for $\eta_{\alpha, \beta} = \alpha^2 + \beta^2$, with

$$
\begin{align*}
\phi_{\alpha}(Dx, z - u) &= \frac{1}{\sqrt{2\pi}} \|Dx - (z - u)\|^2, \\
\psi_{\beta}(u) &= \frac{1}{\sqrt{2\pi}} \|u\|^2.
\end{align*}
$$

(2.3)

In the particular case when $f \equiv 0$, stronger results can be found in [35, Theorem 2], including theoretical guarantees on the convergence rate.

A Gibbs sampler can be used to draw samples from (1.4), approximating the target distribution (1.2). The sampler successively draws one sample from each conditional distribution

$$
\begin{align*}
\pi_\alpha(x | z, u) &\propto \exp \left( -h(x) - f(x) - \phi_{\alpha}(Dx, z - u) \right), \\
\pi_\alpha(z | v, u) &\propto \exp \left( -g(z) - \phi_{\alpha}(v, z - u) \right), \\
\pi_{(\alpha, \beta)}(u | v, z) &\propto \exp \left( -\phi_{\alpha}(v, z - u) - \psi_{\beta}(u) \right),
\end{align*}
$$

(2.4)–(2.6)

where $v = Dx$. The SPA Gibbs sampler associated with (2.4)–(2.6) is thus given by

$$
\begin{align*}
\text{let } z^{(0)} \in G \text{ and } u^{(0)} \in G \\
\text{for } t = 0, 1, \ldots, T
\begin{cases}
\begin{align*}
x^{(t)} &\sim \pi_\alpha(x | z^{(t)}, u^{(t)}), \\
v^{(t)} &\sim \pi_\alpha(z | v^{(t)}, u^{(t)}), \\
z^{(t+1)} &\sim \pi_\alpha(z | v^{(t)}, u^{(t)}), \\
u^{(t+1)} &\sim \pi_{(\alpha, \beta)}(u | v^{(t)}, z^{(t+1)}).
\end{align*}
\end{cases}
\end{align*}
$$

(2.7)

2.2 PSGLA within SPA Gibbs sampler

Drawing samples directly from the conditional distributions (2.4)–(2.6) can still be difficult. This is the case for the application described in Section 6. To overcome this issue, samples can be drawn using appropriate transition kernels.

Let $K_{x,\alpha}(x, z, u)$, $K_{z,\alpha}(Dx, z, u)$ and $K_{u,(\alpha, \beta)}(Dx, z, u)$ be transition kernels with invariant distributions $\pi_\alpha(x | z, u)$, $\pi_\alpha(z | x, u)$, and $\pi_{(\alpha, \beta)}(u | x, z)$, respectively (resp.). A Metropolis-Hastings transition kernel is a classical choice for $K_{x,\alpha}$, $K_{z,\alpha}$ and $K_{u,(\alpha, \beta)}$ to exactly draw samples from (1.4), leading to a conditional Metropolis-Hastings sampler [18]. Appropriate proposals include Langevin-based kernels, such as the Moreau-Yosida unadjusted Langevin algorithm (MYULA) [16] and the proximal stochastic gradient Langevin algorithm (PSGLA) [29]. These kernels can handle differentiable and non-differentiable potential functions simultaneously, and have been shown suitable to address high dimensional problems. To avoid the extra cost of the accept-reject step, approximate sampling can be considered by using unadjusted kernels, as in [16]
(i.e., omitting the accept-reject step). This approach is adopted in the following. Deriving non-asymptotic convergence bounds and analyzing the bias between the target distribution and (1.4) is, however, beyond the scope of this paper.

The choice of suitable transition kernels will be instrumental to design a distributed Gibbs sampler when direct sampling from the conditional distributions is difficult. Technical assumptions on the functions in (1.2), described in Section 4, will ensure the transition kernels to be amenable to a distributed implementation, adapted to the specific structure of the distributions $\pi_\alpha (x \mid z, u)$, $\pi_\alpha (z \mid x, u)$ and $\pi_{\alpha, \beta} (u \mid x, z)$. In particular, for the proposed distributed SPA-Gibbs sampler, we consider the case when any transition kernel can be used to sample $(z, u)$, while $x$ is sampled using one PSGLA iteration. To this aim, we assume that $h$ is $\lambda_h$-Lipschitz-differentiable. In addition, for the sake of simplicity, $\phi_\alpha$ and $\psi_\beta$ are assumed to be $\ell_2$ norms as in (2.3), so that $u$ can be directly sampled from its conditional distribution (i.e., Gaussian distribution). In this case, (2.7) reads

\[
\begin{align*}
\text{let } & x^{(0)} \in H, \ v^{(0)} = Dx^{(0)}, (z^{(0)}, u^{(0)}) \in G^2, \text{ and } \gamma \in [0, \lambda_h + \|D\|^2/\alpha^2]^{-1}, \\
\text{for } & t = 0, 1, \ldots, T \\
& x^{(t+1)} = \text{prox}_{\gamma f} \left( x^{(t)} - \gamma \nabla h(x^{(t)}) - \gamma D^* \left( \nabla \phi_\alpha (\cdot, z^{(t)} - u^{(t)})(v^{(t)}) + \sqrt{2\gamma} w^{(t)} \right) \right), \\
& v^{(t+1)} = Dx^{(t+1)}, \\
& z^{(t+1)} \sim K_{z,\alpha} (v^{(t+1)}, z^{(t)}, u^{(t)}), \\
& u^{(t+1)} \sim N \left( \frac{\beta^2}{\alpha^2 + \beta^2} (z^{(t+1)} - v^{(t+1)}), \frac{\alpha^2 + \beta^2}{\alpha^2 + \beta^2} I \right), \\
\end{align*}
\]

where $D^*$ denotes the adjoint of $D$, and $(w^{(t)})_{1 \leq t \leq T}$ is a sequence of independent and identically distributed (i.i.d) standard Gaussian random variables in $H$.

## 3 Proposed distributed Gibbs sampler in a nutshell

This work aims to design an efficient distributed version of the SPA Gibbs sampler (2.8) over multiple workers by exploiting the hypergraph structure of $D$. In this section, we give a first overview of the main ideas underlying the proposed distributed sampler. The full splitting model is described in Section 4, and the associated distributed Gibbs sampler is given in Section 5.

### 3.1 A hypergraph structure to better communicate

We consider a localized linear operator $D$ in (1.2), meaning that couplings between latent parameters are localized. This is for instance the case when $D$ is block-sparse, see Figure 1. Such structure is not necessary to design the proposed distributed algorithm, but will be instrumental in practice to reduce the communications between workers for an efficient implementation of the algorithm. Linear operators $D$ with block-sparse structures are encountered in multiple inverse
imaging problems, corresponding to, e.g., a deconvolution or inpainting measurement operator. Block-sparse operators also appear when considering local priors such as the TV norm.

The structure of the matrix $D$ can be described with a binary matrix that can be interpreted as the adjacency matrix of a hypergraph. A hypergraph is a generalization of a graph, in which an edge can join any finite number of vertices. Formally, an undirected hypergraph $H$ is a pair $(x, e)$ made of vertices $x = (x_n)_{1 \leq n \leq N}$ and hyperedges $e = (e_m)_{1 \leq m \leq M}$, where each hyperedge is a set of connected vertices.

The main idea is to distribute over $K$ workers the computation of various quantities in algorithm (2.8), related to distribution (1.4), and in particular the product $Dx$. To this aim, the variable $x$ is divided into $K$ blocks $(x_k)_{1 \leq k \leq K}$, each assigned to a single worker. The hyperedges will characterize the necessary communications between the workers. A worker $k \in \{1, \ldots, K\}$, that stores $x_k$, may need to access some coefficients of $x_{k'}$ stored on another worker $k' \in \{1, \ldots, K\} \setminus \{k\}$ to carry out its computations. Note that the operator $D = (D_{m,n})_{1 \leq m \leq M, 1 \leq n \leq N}$ will be split once and for all in an adequate manner over the $K$ workers as well, but will not need to be communicated between the workers.

Similarly to the vertices, the quantity $v = (v_m)_{1 \leq m \leq M}$ computed in (2.8) will also be distributed among the $K$ workers. To this aim, each hyperedge $m \in \{1, \ldots, M\}$ is associated to a worker denoted by $k_m \in \{1, \ldots, K\}$. Then, each subpart $v_m$ will be computed and stored on worker $k_m$. In practice, only the vertices $x_n$ that correspond to non-zero blocks $D_{m,n}$ in $D$ will be necessary to compute $v_m$. In addition, only a subset of $x_n$ will be used to compute $v_m$, but will not be used to compute any other $v_{m'}$. This is illustrated in Figure 1. Finally, the worker $k = k_m$ that is in charge of the computation of $v_m$ will also store $x_k$ to reduce communications.

Communications between workers will be necessary when the worker $k_m$ requires a subpart of $x_{k'}$ stored on another worker $k' \neq k_m$ to carry out the computation of $v_m$. In the particular case when $D$ is block-sparse, the communication cost will remain small as long as the subpart of $x_{k'}$ required by worker $k_m$ remains small.

### 3.2 Conditions for an efficient distributed implementation

As previously mentioned, although the proposed approach detailed in Section 4 and Section 5 is very general, some particular structures of matrix $D$ can ensure limited communications, which are often the bottleneck of distributed methods.

Computing costs of algorithms can roughly be divided into three terms. First, a computation term $\tau_{\text{flop}}$ reflects the time to perform a single floating point operation. Second, a communication term $\tau_{\text{bandwidth}}$, defined as the inverse of the communication bandwidth, quantifies the time necessary to send a single value. Third, a latency term $\tau_{\text{latency}}$ represents the cost incurred by establishing a communication. In practice, $\tau_{\text{flop}} \ll \tau_{\text{bandwidth}} \ll \tau_{\text{latency}}$. This implies that the
number of communications and the size of the messages need to be sufficiently small to ensure the overall communication time to be negligible compared to the computation time.

Consequently, to ensure a higher efficiency of the proposed distributed SPA Gibbs sampler, both in terms of computations and communications, we can identify two conditions on the structure of $D$. These conditions are typically satisfied when $D$ is block-sparse, as the operators considered in Section 6.

The first condition consists in ensuring that, each worker only needs to communicate with a small amount of other workers compared to the total number of workers $K$.

The second condition is that the quantity of variables that two communicating workers need to exchange remains small compared to the variables already stored on each worker. In particular, for two workers $(k,k') \in \{1,\ldots,K\}^2$, $k \neq k'$, the subpart of $x_k$ that need to be sent from $k$ to $k'$ needs to be small compared to $x_k$ and $x_{k'}$.

4 Model and hypergraph structure

This section provides the notation and a description of the hypergraph model used to define the proposed distributed SPA Gibbs sampler. It gives a formal description of the proposed distributed computing strategy, which exploits the structure of the linear operator $D$ and a separability assumption on the functions involved in (1.4). Notation are summarized in Tables 1 and 2, and illustrated on a simple example in Figure 2.

4.1 Hypergraph structure of the model

Let $H = \mathbb{R}^N$ be such that $H = H_1 \times \ldots \times H_N$, where for every $n \in \{1,\ldots,N\}$, $H_n = \mathbb{R}^{N_n}$ and $N = \sum_{n=1}^{N} N_n$. An element of $H$ is denoted by $x = (x_n)_{1 \leq n \leq N}$, where, for every $n \in \{1,\ldots,N\}$, $x_n \in H_n$. Similarly, let $G = \mathbb{R}^M$ be such that $G = G_1 \times \ldots \times G_M$, where for every $m \in \{1,\ldots,M\}$, $G_m = \mathbb{R}^{M_m}$ and $M = \sum_{m=1}^{M} M_m$. Let $v = (v_m)_{1 \leq m \leq M}$ be an element of $G$ such that, for every $m \in \{1,\ldots,M\}$, $v_m \in G_m$.

The distribution of the problem over the different workers will follow the topology of a hypergraph encoded by the structure of $D$. To this aim, we assume that $D$ holds some block separability structure, and consider the following hypergraph model.

Model 4.1 Let $D : H \to G$ be such that $D = (D_{m,n})_{1 \leq m \leq M, 1 \leq n \leq N}$, with, for every $(m,n) \in \{1,\ldots,M\} \times \{1,\ldots,N\}$, $D_{m,n} : H_n \to G_m$. Let $\mathcal{H} = (x,e)$ be the hypergraph associated with $D$, with $N$ vertices $x$, and $M$ hyperedges denoted by $e = (e_m)_{1 \leq m \leq M}$, such that

$$\forall m \in \{1,\ldots,M\} \quad e_m = \{ n \in \{1,\ldots,N\} \mid D_{m,n} \neq 0 \},$$

(4.1)
where \( 0_{G_m \times H_n} \) is the null element from \( G_m \) to \( H_n \).

The hyperedges of \( \mathcal{H} \), i.e., the connections between vertices, are described by the topology of \( D \). Precisely, the \( M \) rows of \( D \) represent the \( M \) hyperedges of \( \mathcal{H} \), and the \( N \) columns represent the \( N \) vertices of \( \mathcal{H} \). Hence, as described in Theorem 4.1, for each \( m \in \{1,\ldots,M\} \), the hyperedge \( e_m \) links nodes \( (x_n) \) if \( D_{m,n} \neq 0_{G_m \times H_n} \). Hence, \( D \) can be seen as a weighted incidence matrix associated with \( \mathcal{H} \). In this context, any variable in \( G \) can be seen as a hyperedge weight of the hypergraph \( \mathcal{H} \).

According to Theorem 4.1, the computation of \( v = Dx \) in Algorithm (2.8) can be decomposed and computed block-wise as

\[
v = (v_m)_{1 \leq m \leq M} \quad \text{with} \quad (\forall m \in \{1,\ldots,M\}) \quad v_m = \sum_{n \in e_m} D_{m,n} x_n. \tag{4.2}
\]

In (4.2), for every \( m \in \{1,\ldots,M\} \), only the non-zero blocks of \( D \) are taken into account, i.e., only using the vertices contained in the hyperedge \( e_m \). In the particular case when all hyperedges are disjoint (i.e., disconnected hypergraph), a very simple distributed algorithm would distribute the computation of the quantities \( v_m \) on independent workers. This is not the case in general, and hyperedges \( e_m \) can share some vertices \( x_n \). A distributed algorithm will thus need to carry out communications between some workers. The next section exploits the hypergraph model described here to distribute the computation of \( Dx \), while limiting the communication cost.

### 4.2 Distribution of the hypergraph nodes over the workers

Let \( K \in \mathbb{N}^* \), \( K \leq N \), be the number of workers available to the user to parallelize the algorithm. On each worker \( k \in \{1,\ldots,K\} \), we store a subpart \( x_k \) of the nodes \( x \) of the hypergraph \( \mathcal{H} \). The pattern for the distribution of the nodes \( x \) over the \( K \) workers will be driven by the topology of the matrix \( D \), as explained in Section 4.1. This pattern needs to be fixed before designing the algorithm. Note that the blocks \( (D_{m,n})_{1 \leq m \leq M, n \in e_m} \) of \( D \) will be distributed over the workers as well, but will not require to be communicated between the workers. In particular, we provide below the notation to split the hypergraph (i.e., the operator \( D \)) over the \( K \) workers. The notation for the splitting of the nodes and the hyperedge weights are illustrated in Figure 2 for a simple example of a block-sparse matrix \( D \).

#### 4.2.1 Distributing the hypergraph over the workers

Let \( (\mathcal{V}_k)_{1 \leq k \leq K} \) be the partition of the set of vertex indices \( \{1,\ldots,N\} \) such that, for every \( k \in \{1,\ldots,K\} \), \( \mathcal{V}_k \) is the set of vertex indices handled on worker \( k \), and for every \( x \in \mathcal{H}, x_k = (x_n)_{n \in \mathcal{V}_k} \). Recall that \( k_m \in \{1,\ldots,K\} \) denotes the worker associated with the \( m \)-th hyperedge.
The association of hyperedges to workers is a choice left to the user. Then, we must have $e_m \cap V_{k_m} \neq \emptyset$, i.e., worker $k_m$ must contain at least one vertex belonging to hyperedge $e_m$.

The set of all the hyperedges associated with worker $k$ is denoted by

$$E_k = \{ m \in \{1, \ldots, M \} \mid k_m = k \}. \tag{4.3}$$

Note that $(E_k)_{1 \leq k \leq K}$ defines a partition of $\{1, \ldots, M\}$ over the $K$ workers. Note that optimizing the configuration for a specific operator $D$ is on its own a resource allocation problem [20] that is out of the scope of the current work. Using notation (4.3), for any element $v \in G$, we can denote by $v_k = (v_m)_{m \in E_k}$ the hyperedge weights stored on worker $k$.

We also define $E_k$, the set of hyperedge indices only containing vertices stored on worker $k$, as

$$E_k = \{ m \in \{1, \ldots, M \} \mid n \in e_m \iff n \in V_k \}. \tag{4.4}$$

The sets $(E_k)_{1 \leq k \leq K}$ identify the rows of $D$ whose non-zero elements are multiplied with vertices that are stored on a single worker, i.e., the rows that can be used with no communication between two different workers. In contrast, the computation of $v_m$ for $m \in \overline{E}_k \setminus E_k$ will call for communications.

### 4.2.2 Hyperedges overlapping over workers ensuring communications

The hyperedges overlapping over multiple workers will require vertices stored on different workers to be communicated between each other. They correspond to hyperedges $e_m$, with $m \in \{1, \ldots, M\} \setminus (\bigcup_{k=1}^{K} E_k)$.

For a fixed hyperedge $e_m$, let $W_m \subset \{1, \ldots, K\} \setminus \{k_m\}$ be the set of workers other than $k_m$ containing vertices $x_n$ belonging to the same hyperedge $e_m$, i.e.,

$$W_m = \{ k \in \{1, \ldots, K\} \setminus \{k_m\} \mid \exists n \in e_m \text{ such that } n \in V_k \}.$$  

Using this notation, we can give an equivalent definition of (4.4) as, for every $k \in \{1, \ldots, K\}$, $E_k = \{ m \in \{1, \ldots, M\} \mid k_m = k \text{ and } W_m = \emptyset \}$ (i.e., no overlap, no communication). For completeness, we also introduce the notation $W_m = k_m \cup W_m$.

For every $m \in \{1, \ldots, M\} \setminus (\bigcup_{k=1}^{K} E_k)$, workers $k' \in W_m$ will need to send some vertex values from $x_{k'}$ to the worker $k_m$ so that it can compute $v_m$, as described in (4.2). For every $k \in \{1, \ldots, K\}$, we denote by $R_k \subset \{1, \ldots, K\} \setminus \{k\}$ the set of workers $k'$ from which worker $k$ receives vertices. These workers store vertices belonging to a hyperedge of index $m \in \{1, \ldots, M\}$ such that $k_m = k$ but that are not stored on worker $k$. Similarly, we denote by $S_k \subset \{1, \ldots, K\} \setminus \{k\}$ the set of workers $k'$ to which worker $k$ is sending vertices, i.e., all the workers $k'$ such that $k \in R_{k'}$.

Communications will occur through hyperedges connecting different workers. For every $k' \in R_k$, let

$$E_{(k,k')} = \{ m \in \{1, \ldots, M\} \mid k_m = k \text{ and } k' \in W_m \},$$

11
be the set of hyperedges containing the vertices from worker $k'$ that will be needed by worker $k$ to compute $v_m$, so that $E_{k'} = \{ k' \in \{1, \ldots, K\} \setminus \{k\} \mid \exists m \in E_{(k,k')} \}$. Thus, the set of all the hyperedges that will carry out some communication to worker $k$ is

$$E_R_k = \bigcup_{k' \in R_k} E_{(k,k')} = \{ m \in \{1, \ldots, M\} \mid k_m = k \text{ and } \mathcal{W}_m \neq \emptyset \}. $$

As a result, $E_k$ and $E_R_k$ form a partition of $E_k$, where $E_k$ corresponds to the set of weights $v_m$ that can be computed locally on worker $k$ (i.e., hyperedges inducing no communication), and $E_R_k$ corresponds to the weights $v_m$ that necessitate vertices provided by other workers, and therefore communications.

Each hyperedge $e_m$ potentially sends vertices towards the corresponding worker $k_m$. To identify vertices that are communicated between workers, let

$$V_{(k,k')} = \{ n \in V_{k'} \mid \exists m \in E_{(k,k')} \text{ such that } n \in e_m \} $$

be the indices of vertices that are received by worker $k$ from $k'$. The set of vertices sent to worker $k_m$ by all workers in $\mathcal{W}_m$ is denoted by $V_{\mathcal{W}_m} = \bigcup_{k' \in \mathcal{W}_m} V_{(k,m')}$. For completeness, we also introduce $V_{\overline{\mathcal{W}}_m} = \bigcup_{k' \in \overline{\mathcal{W}}_m} V_{(k,m')}$, the set of all the vertex indices necessary to perform computations associated with the $m$-th hyperedge.

For every $x \in H$, we denote by $x_{(k,k')} = (x_n)_{n \in V_{(k,k')}}$ the vertices communicated from worker $k'$ to worker $k$ through some hyperedge $e_m$ such that $k_m = k$. For every $m \in \{1, \ldots, M\} \setminus (\bigcup_{k=1}^K E_k)$, let $x_{\overline{\mathcal{W}}_m} = (x_n)_{n \in V_{\overline{\mathcal{W}}_m}}$ be the concatenation of vertices stored on worker $k_m$ and vertices sent from all workers $k' \in \mathcal{W}_m$ to worker $k_m$ (i.e., all the vertices such that $D_{m,n} \neq 0$).

**Remark 4.2** Let $k \in \{1, \ldots, K\}$. For every $k' \in S_k$, since $V_{(k,k')}^{-}$ denotes the set of vertices that need to be communicated from worker $k'$ to worker $k$, the (reciprocal) set $V_{(k',k)} \subset V_k$ corresponds to the set of vertices that will be sent back from worker $k$ to $k'$ in the distributed implementation of the adjoint operator $D^*$. Similarly, with $E_{(k,k')}^{-}$ the set of hyperedges containing vertices that will be communicated from worker $k'$ to worker $k$, the set $E_{(k',k)} \subset \{1, \ldots, M\}$ corresponds to the hyperedge indices containing vertices that will be sent back from worker $k$ to worker $k'$.

### 4.2.3 Splitting of $D$ over the workers

Eventually, we will introduce some notation to split the matrix $D$ over the workers. In practice, the input space $G$ and the output space $H$ are partitioned using the sets $(E_k)_{k \in \{1, \ldots, K\}}$ and $(V_k)_{k \in \{1, \ldots, K\}}$, respectively. For every $k \in \{1, \ldots, K\}$, let $G_k = \times_{m \in \overline{V}_k} G_m$ and $H_k = \times_{n \in V_k} H_n$ such that $G = \times_{1 \leq k \leq K} G_k$ and $H = \times_{1 \leq k \leq K} H_k$.

For every $k \in \{1, \ldots, K\}$, the subparts $v_k$ of $Dx$ that belong to $G_k$ are stored on worker $k$. The subparts of $D$ acting on $H_k$ only are associated with vertices in $V_k$. For every $m \in E_k$, they will be
denoted by $D_{m,k} = (D_{m,n})_{n \in V_k}$. These subparts of $D$ are involved in purely local computations only. Similarly, for every $k' \in R_k$, for every $m \in E_{(k,k')}$, the subparts of $D$ acting on vertices $V_{(k,k')}$ will be denoted by $D_{m,(k,k')} = (D_{m,n})_{n \in V_{k'} \cup V_{(k,k')}}$. In addition, for every $m \in E_{R_k}$, we will denote by $D_{m, V_W} = (D_{m,n})_{n \in V_{W_m}}$ the subparts of $D$ acting on vertices stored either on worker $k_m$ or sent to worker $k_m$ by all other workers $k' \in W_m$. Hence

$$v_m = \begin{cases} D_{m,k}x_k, & \text{if } m \in E_k, \\ D_{m, V_W}x_{V_W}, & \text{otherwise.} \end{cases}$$

As a result, the subparts of $D$ which must be stored on worker $k$ are the $D_{m,k}$, for every $m \in E_k$, and the $D_{m, V_W}$, for every $m \in E_{R_k}$.

The notation given above are summarized in Table 1 and Table 2. A simple example for block-sparse matrices is also provided below, and illustrated in Figure 2. This example will be reused later on to illustrate the communications between the workers (see Figure 3).
Figure 2: Distribution of vertices (horizontal lines) and hyperedges (vertical lines) of the hypergraph over $K = 3$ workers, for a block-sparse matrix $D$ corresponding to Theorem 4.3.

Example 4.3 To illustrate some of the notation introduced above, consider the block-sparse matrix $D$ shown in Figure 2, with $K = 3$ workers. For the vertices, $V_1$ (resp. $V_2$ and $V_3$) contains the vertices handled on worker $k = 1$ (resp. $k = 2$ and $k = 3$). In addition, $V_{(1,2)}$ (resp. $V_{(2,3)}$) identifies the vertices of $x_2$ (resp. $x_3$) that will need to be communicated to worker $k = 1$ (resp. $k = 2$). For the hyperedges, $E_1$ (resp. $E_2$ and $E_3$) identifies the hyperedge weights stored on worker $k = 1$ (resp. $k = 2$ and $k = 3$). The set $E_1$ (resp. $E_2$ and $E_3$) identifies the hyperedges fully contained in worker $k = 1$ (resp. $k = 2$ and $k = 3$). Hyperedges in $E_1$ (resp. $E_2$ and $E_3$) identify the vertices that will be communicated from worker $k = 2$ to worker $k = 1$ (resp. $k = 2$ to $k = 3$). In this example, only the blocks $(D_{m,k})_{m \in E_1}$ (resp. $(D_{m,(1,2)})_{m \in E_{(1,2)}}$ and $(D_{m,(2,3)})_{m \in E_{(2,3)}}$) are non-zero (e.g., convolution operator). Blocks $(D_{m,1})_{m \in E_1}$ (resp. $(D_{m,(1,2)})_{m \in E_{(1,2)}}$) and $(D_{m,3})_{m \in E_3}$ are stored on worker $k = 1$ (resp. $k = 2$ and $k = 3$). For worker $k = 1$ (resp. $k = 2$ and $k = 3$), workers sending vertex values to this worker is given by $\mathcal{R}_1 = \{2\}$ (resp. $\mathcal{R}_2 = \{3\}$ and $\mathcal{R}_3 = \emptyset$). Similarly, workers receiving vertex values from this worker is given by $\mathcal{S}_1 = \emptyset$ (resp. $\mathcal{S}_2 = \{1\}$ and $\mathcal{S}_3 = \{2\}$). For every $m \in E_1$ (resp. $E_2$ and $E_3$), $k_m = 1$ (resp. $k_m = 2$ and $k_m = 3$). For every $m \in E_{(1,2)}$, $\mathcal{V}_m = \mathcal{R}_1 = \mathcal{S}_2 = \{2\}$, and for every $m \in E_{(2,3)}$, $\mathcal{V}_m = \mathcal{R}_2 = \{3\}$. Finally, for every $m \in E_{(1,2)} = E_{(1,2)}$ (resp. $E_{(2,3)} = E_{(2,3)}$), $D_{m,W_m} = (D_{m,n})_{n \in V_1 \cup V_{(1,2)}}$ (resp. $D_{m,W_m} = (D_{m,n})_{n \in V_2 \cup V_{(2,3)}}$). These last notation correspond to the orange rectangles displayed in Figure 1. In this example, $E_{R_3} = \emptyset$. 
4.3 Separability assumptions

We can now give assumptions that will be used to design an efficient distributed SPA Gibbs sampler. Precisely, we will consider distribution (1.4), where the functions are assumed to satisfy some separability conditions.

**Assumption 4.4**

(i) \( h: \mathcal{H} \rightarrow \mathbb{R} \) is \( \lambda_h \)-Lipschitz differentiable, with \( \lambda_h > 0 \).

(ii) \( h: \mathcal{H} \rightarrow \mathbb{R} \) and \( f: \mathcal{H} \rightarrow ]-\infty, +\infty[ \) are additively separable on the workers, i.e.

\[
(\forall x \in \mathcal{H}) \quad h(x) = \sum_{k=1}^{K} h_k(x_k), \quad \text{and} \quad f(x) = \sum_{k=1}^{K} f_k(x_k),
\]

where, for every \( k \in \{1, \ldots, K\} \), \( h_k: \mathcal{H}_k \rightarrow \mathbb{R} \) and \( f_k: \mathcal{H}_k \rightarrow ]-\infty, +\infty[ \).

(iii) \( g: \mathcal{G} \rightarrow ]-\infty, +\infty[ \) is additively separable, i.e.

\[
(\forall v \in \mathcal{G}) \quad g(v) = \sum_{m=1}^{M} g_m(v_m),
\]

with, for every \( m \in \{1, \ldots, M\} \), \( g_m: \mathcal{G}_m \rightarrow ]-\infty, +\infty[ \).

(iv) \( \phi_\alpha \) and \( \psi_\beta \) are given by

\[
(\forall (v, u) \in \mathcal{G}^2) \quad \phi_\alpha(v, u) = \sum_{m=1}^{M} \phi_{m,\alpha}(v_m, u_m) = \sum_{m=1}^{M} \frac{1}{2\alpha^2} \|v_m - u_m\|^2,
\]

\[
\psi_\beta(u) = \sum_{m=1}^{M} \psi_{m,\beta}(u_m) = \sum_{m=1}^{M} \frac{1}{2\beta^2} \|u_m\|^2.
\]

The above assumptions can be rewritten to highlight the separability over the workers. Indeed, there exists a permutation \( \varrho: \mathcal{G} \rightarrow \mathcal{G} \) such that

\[
(\forall x \in \mathcal{H}) \quad D x = \left( (D_{m,n})_{1 \leq n \leq N} x \right)_{1 \leq m \leq M} = \varrho \left( \left( \left( D_{m,k} \right)_{m \in \mathcal{E}_k} x_k \right) \left( \left( D_{m,m}^{-1} x_m \right)_{m \in \mathcal{E}_k} \right)_{1 \leq k \leq K} \right).
\]

Reorganizing the vertex indices emphasizes the distinction between local computations and those for which communications are required. For every \((u, v) \in \mathcal{G}^2\) and \( x \in \mathcal{H} \), (4.7), (4.8) and (4.9)
Table 2: Notation used for the variables involved in the distributed algorithm.

| Notation | Definition |
|----------|-----------|
| \(x = (x_n)_{1 \leq n \leq N} \in H\) | Vertex values |
| \(D : H \to G\) | Linear operator defining the hypergraph structure |
| \(u, v \in G\) | Hyperedge weights |
| \(v = (v_m)_{1 \leq m \leq M} \in G\) | 
| \(v_k \in G_k\) | Hyperedge weights stored on worker \(k\) |
| \(u, v \in G\) | Vertex values stored on worker \(k\) |
| \(D_{m,k} : H_k \to G_m\) | Subpart of \(D\) stored on worker \(k\), associated with hyperedges containing vertices only on worker \(k\) |
| \(D_{m,(k,k')} : \times_{n \in V_{(k,k')}} H_n \to G_m\) | Subpart of \(D\) stored on worker \(k\), associated with hyperedges containing vertices overlapping workers \(k\) and \(k'\) |
| \(D_{m,(k,k')} : \times_{n \in V_{(k,k')}} H_n \to G_m\) | Concatenation of vertex values stored on worker \(k_m\), and those sent from all worker \(k' \in \mathcal{W}_m\) to worker \(k_m\) |

\[
g(v) = \sum_{k=1}^{K} \left( \sum_{m \in E_k} g_m(v_m) \right), \tag{4.11}
\]

\[
\phi_\alpha(Dx, u) = \sum_{k=1}^{K} \left( \sum_{m \in E_k} \phi_{m,\alpha}(Dx_m^m, u_m) \right), \tag{4.12}
\]

\[
= \sum_{k=1}^{K} \left( \sum_{m \in E_k} \phi_{m,\alpha}(D_{m,k}x_k^m, u_m) + \sum_{m \in E_{R,k}} \phi_{m,\alpha}(D_{m,(k,k')}x_{(k,k')}^m, u_m) \right), \tag{4.13}
\]

\[
\psi_\beta(u) = \sum_{k=1}^{K} \left( \sum_{m \in E_k} \psi_{m,\beta}(u_m) \right), \tag{4.14}
\]

where \([::m]m\) denotes the \(m\)-th element of its argument.

## 5 Distributed SPA Gibbs sampler

### 5.1 Proposed distributed algorithm

Using the model from the previous section, we are now able to introduce a distributed block-coordinate version of Algorithm (2.8). The term *block-coordinate* is to be interpreted as in the optimization literature, where the variables from (1.4) are divided into \(K\) blocks distributed over the \(K\) workers, and updated in parallel. The proposed distributed algorithm is the following.
**Proposition 5.1** Consider a distribution (1.4) satisfying Theorem 4.4. Assume that the operator $D$ follows Theorem 4.1, and is split over workers $\{1, \ldots, K\}$ such that, for every $k \in \{1, \ldots, K\}$, $(D_{m,k})_{m \in H_k}$ is stored on worker $k$. For every $k \in \{1, \ldots, K\}$, let $x_k^{(0)} \in H_k$, $z_k^{(0)} \in G_k$, and $u_k^{(0)} \in G_k$. Let $(x^{(t)}, z^{(t)}, u^{(t)})_{1 \leq t \leq T}$ be samples generated by the following algorithm:

\[
\begin{align*}
\text{for } k = 1, \ldots, K \\
\text{for } k' \in S_k \\
&\quad \text{Send } (x_n^{(0)})_{n \in V(k',k)} \text{ to worker } k', \\
\text{for } k' \in R_k \\
&\quad \text{Receive } (x_n^{(0)})_{n \in V(k',k)} \text{ from worker } k', \\
&\quad v_k^{(0)} = ((D_{m,k}x_n^{(0)})_{m \in H_k}, (D_{m,W_m}x^{(0)}_{W_m})_{m \in R_k}), \\
\text{for } t = 0, 1, \ldots, T \\
\text{for } k = 1, \ldots, K \\
&\quad (d^{(t)}_{m})_{m \in H_k} = \phi_{m,\alpha}(\cdot, z_m^{(t)} - u_m^{(t)}(v_m^{(t)}))_{m \in H_k}, \\
&\quad \text{for } k' \in R_k \\
&\quad \hat{d}^{(t)}_{(k',k)} = \sum_{m \in H_{k'}} D^*_{m,k}d^{(t)}_m, \text{ Send } \hat{d}^{(t)}_{(k',k)} \text{ to worker } k', \\
&\quad \text{for } k' \in S_k \\
&\quad \text{Receive } \hat{d}^{(t)}_{(k,k')} \text{ from worker } k', \\
&\quad \delta^{(t)}_k = \sum_{m \in H_k} D^*_{m,k}d^{(t)}_m + \sum_{k' \in S_k} \hat{d}^{(t)}_{(k',k)} \\
&\quad x^{(t+1)}_k = \text{prox}_{\gamma f_k}(x^{(t)}_k - \gamma \nabla h_k(x^{(t)}_k) - \gamma \delta^{(t)}_k + \sqrt{2\gamma}w^{(t)}_k), \\
&\quad \text{for } k' \in S_k \\
&\quad \text{Send } (x_n^{(t+1)})_{n \in V(k',k)} \text{ to worker } k', \\
&\quad \text{for } k' \in R_k \\
&\quad \text{Receive } (x_n^{(t+1)})_{n \in V(k',k')} \text{ from worker } k', \\
&\quad v^{(t+1)}_k = \rho \left( (D_{m,k}x^{(t+1)}_n)_{m \in H_k} \right), \\
&\quad z^{(t+1)}_k \sim \mathcal{N}_k(z^{(t)}_k; w^{(t)}_k), \\
&\quad u^{(t+1)}_k \sim \mathcal{N}_k (z^{(t+1)}_k - v^{(t+1)}_k; \frac{\alpha^2 + \beta^2}{\alpha^2 + \beta^2} I), \\
\end{align*}
\]

where $\gamma \in [0, (\lambda_h + \|D\|^2/\alpha^2)^{-1}]$ and, for every $k \in \{1, \ldots, K\}$, $(w_k^{(t)})_{1 \leq t \leq T}$ is a sequence of i.i.d. standard Gaussian random variables in $H_k$. In addition, the transition kernel $\mathcal{K}_{z_k,\alpha}$ has the following
invariant distribution

\[ \pi_{k,\alpha}(z_k | v_k, u_k) \propto \exp \left( - \sum_{m \in E_k} (g_m(z_m) + \phi_{m,\alpha}(v_m, z_m - u_m)) \right) \]  \hspace{1cm} (5.2)

where, for every \( x \in \mathbf{H} \), \( v_k = (v_m)_{m \in E_k} = \rho \left( (D_{m,k} x_k)_{m \in E_k} \right) \).

Then, Algorithm (5.1) is equivalent to Algorithm (2.8).

All the computations described in Algorithm 5.1 are conducted simultaneously on each worker, after the necessary communication phases.

Before giving the proof of this result, we want to emphasize that communications to apply the operator \( D \) and its adjoint \( D^* \) are symmetric: the same workers are involved in communications, but the direction of the communications (send/receive) is reversed; the indices of the subparts that are communicated are not the same. Communications necessary to computations involving \( D \) are received by worker \( k \) from those of \( R_k \); communications necessary to computations involving \( D^* \) are received by worker \( k \) from those of \( S_k \). Figure 3(a) illustrates the communications involved in the application of the operator \( D \) from Theorem 4.3. Figure 3(b) illustrates the communications associated with the application of the adjoint operator \( D^* \).

Proof. Using notation from Section 4, (4.7)–(4.9) in Theorem 4.4 can be directly re-written as in (4.11)–(4.14). Then, fixing \( v = Dx \), the conditional distributions (2.5)-(2.6) can be rewritten as

\[ \pi_{\alpha}(z | v, u) = \prod_{k=1}^{K} \pi_{k,\alpha}(z_k | v_k, u_k), \]  \hspace{1cm} (5.3)

\[ \pi_{(\alpha,\beta)}(u | v, z) = \prod_{k=1}^{K} \pi_{k,(\alpha,\beta)}(u_k | v_k, z_k), \]  \hspace{1cm} (5.4)

respectively, where, for every \( k \in \{1, \ldots, K\} \),

\[ \pi_{k,\alpha}(z_k | v_k, u_k) \propto \exp \left( - \sum_{m \in E_k} (g_m(z_m) + \phi_{m,\alpha}(v_m, z_m - u_m)) \right), \]  \hspace{1cm} (5.5)

\[ \pi_{k,(\alpha,\beta)}(u_k | v_k, z_k) \propto \exp \left( - \sum_{m \in E_k} (\phi_{m,\alpha}(v_m, z_m - u_m) - \psi_{m,\beta}(u_m)) \right), \]  \hspace{1cm} (5.6)

\[ = \mathcal{N}\left( \frac{\beta^2}{\alpha^2 + \beta^2} (z_k - v_k), \frac{\alpha^2}{\alpha^2 \beta^2 - 1}\right), \]

for \( v_k = (v_m)_{m \in E_k} = ([Dx]_m)_{m \in E_k} \). Hence, for every iteration \( t \in \{0, \ldots, T\} \) of the Algorithm (2.7), and for every worker \( k \in \{1, \ldots, K\} \), the sampling of \( z_k^{(t+1)} \) and \( u_k^{(t+1)} \) only requires
partial information from \((v_k^{(t+1)}, u_k^{(t)})\) and \((v_k^{(t+1)}, z_k^{(t+1)})\), respectively. Algorithm (2.8) is thus equivalent to

\[
\begin{align*}
\text{for } t = 0, 1, \ldots, T & \quad \text{prox}_{\gamma_f} \left( x^{(t)} - \gamma \nabla h(x^{(t)}) - \gamma D^* \left( \nabla \phi_k(\cdot, z^{(t)} - u^{(t)})((v_m^{(t)})_{m \in E_k}) \right) \right) \\
\text{for } k = 1, \ldots, K & \quad \begin{cases} \\
\begin{pmatrix} v_k^{(t+1)} = ([Dx]^{(t+1)}_{m \in E_k}) \\
z_k^{(t+1)} \sim \mathcal{K}_{z_k, \alpha}(v_k^{(t+1)}, z_k^{(t)}, u_k^{(t)}) \\
u_k^{(t+1)} \sim \mathcal{N}\left( \frac{\beta^2}{\alpha + \beta^2}(z_k^{(t+1)} - v_k^{(t+1)}), \frac{\alpha^2 + \beta^2}{\alpha + \beta^2} \mathbf{I} \right)
\end{cases}
\end{align*}
\]

(5.7)

where, for every \(k \in \{1, \ldots, K\}\), \(\mathcal{K}_{z_k, \alpha}(Dx|_{m \in E_k}, z_k, u_k)\) is a transition kernel with invariant distribution \(\pi_{k, \alpha}(z_k | [Dx]_{m \in E_k}, u_k)\).

It remains to show that, for every \(t \in \{0, \ldots, T\}\), the computation of \(x^{(t+1)}\) can be parallelized over \(k \in \{1, \ldots, K\}\). Since \((V_k)_{1 \leq k \leq K}\) is a partition of \(\{1, \ldots, N\}\), then, for every \(k \in \{1, \ldots, K\}\), \(x_k^{(t+1)}\) will be given by the \(k\)-th vertex values of the random variable generated from the PSGLA transition kernel.

The gradient of (4.13) with respect to \(x\) is required to compute \((x_k^{(t+1)})_{1 \leq k \leq K}\). Using the chain rule on \(x \in \mathbf{H} \mapsto (\phi_\alpha(\cdot, z - u) \circ D)(x)\), for \((z, u) \in \mathbf{G}^2\) fixed, we obtain, for every \(x \in \mathbf{H}\),

\[
\nabla_x \phi_\alpha(Dx, z - u) = \nabla_x (\phi_\alpha(\cdot, z - u) \circ D)(x) = D^* \nabla \phi_\alpha(\cdot, z - u)(Dx).
\]

(5.8)

According to (4.13), for every \((v, z) \in \mathbf{G}^2\), we have

\[
\nabla_v \phi_\alpha(v, z) = \varrho \left( \left( \left( \phi_{m, \alpha}(v_m, z_m) \right)_{m \in E_k} \right) \right),
\]

(5.9)

where \(\varrho\) is the permutation operator defined in (4.10). For every \(k \in \{1, \ldots, K\}\), every \(x \in \mathbf{H}\) and \((z, u) \in \mathbf{G}^2\), let \((d_m)_{m \in E_k}\) be defined as

\[
d_m := \phi'_{m, \alpha}(\cdot, z_m - u_m)(v_m) = \begin{cases} \\
\phi'_{m, \alpha}(\cdot, z_m - u_m)(D_{m,k}x_k), & \text{if } m \in E_k, \\
\phi'_{m, \alpha}(\cdot, z_m - u_m)(D_{m,W_m}x_{W_m}), & \text{if } m \in E_{R_k}.
\end{cases}
\]

(5.10)

Finally,

\[
\nabla_x \phi_\alpha(Dx, z - u) = D^* \varrho \left( \left( (d_m)_{m \in E_k} \right)_{1 \leq k \leq K} \right).
\]

(5.11)

To extract the \(k\)-th block from this gradient, (5.11) can be decomposed as follows:

\[
\delta_k := \left[ \nabla_x \phi_\alpha(Dx, z - u) \right]_k = \sum_{m \in E_k} D_{m,k}d_m + \sum_{k' \in S_k} \sum_{m \in E_{(k,k')}} D_{m,k}d_m.
\]

(5.12)
Figure 3 illustrates this for the simple example described in Figure 2.

The second term in (5.12) gathers information received from the set of neighbours $S_k$ of worker $k$ that is necessary to computations involving $D^*$. Then, using the notation

$$(\forall k' \in S_k) \quad \tilde{d}_{(k,k')} = \sum_{m \in E_{(k,k')}} D^*_{m,k} d_m,$$

we obtain

$$\delta_k = \sum_{m \in E} D^*_{m,k} d_m + \sum_{k' \in S_k} \tilde{d}_{(k,k')}.$$  (5.14)

where, for every $k' \in S_k$, $\tilde{d}_{(k,k')}$ is computed on worker $k'$ and communicated to worker $k$ to form $\delta_k$. Therefore the computations performed on a given worker $k \in \{1, \ldots, K\}$ include the computation of local gradients $(D^*_{m,k} d_m)_{m \in E_k}$ as well as the gradients to be sent to neighbours, i.e., $(\tilde{d}_{(k',k)})_{k' \in R_k}$. Note that the role of the sets $R_k$ and $S_k$ is exchanged when the adjoint operator $D^*$ is considered.

Using the above notation, and using [1, Prop. 24.11] on the additively separable functions $h$ and $f$, leads to the conclusion that (5.1) is equivalent to (5.7), and therefore to Algorithm (2.8). \(\square\)

(a) Computation of $Dx$  
(b) Computation of $D^*d$

Figure 3: Illustration of the computation of (a) $Dx$ for $x \in H$, and (b) $D^*d$ for $d \in G$. $D$ corresponds to the block-sparse matrix from Figure 2, and $D^*$ to its adjoint. The yellow, green and blue colours identify how $D$, $D^*$, $x$ and $d$ are distributed over the workers $k = 1$, $k = 2$, and $k = 3$, respectively. Communications are illustrated by coloured arrows over subparts of $x$ and $d$, respectively. Continuous coloured lines on $x$ and $d$ identify vertices and hyperedge weights required to perform local computations. The corresponding blocks in $D$ and $D^*$ are emphasized accordingly. Vertices and hyperedge weights to be sent from a worker to another are delineated in dashed lines, with the corresponding blocks within $D$ and $D^*$ highlighted accordingly. (a) Vertices are on columns, and hyperedges on rows. (b) Vertices are on rows, and hyperedges on columns. The circled “plus” symbol emphasizes that hyperedge weights are aggregated on the reception workers.

**Example 5.2** Figure 3(a) illustrates the computation of $Dx$, for $x \in H$ for the block-sparse matrix $D$ of Example 4.3, see also Figure 1. Figure 3(b) illustrates the computation of $D^*d$, for $d \in G$.
with some adjoint symmetry. Subparts $V_{(1,2)}$ of $x_2$ and $V_{(2,3)}$ of $x_3$ need to be communicated to workers 1 and 2, respectively, to compute $D \mathbf{x}$. In Figure 3(a), for worker $k = 1$, the quantity $(D_{m,W_m} x_{W_m})_{m \in \mathcal{E}_{V_1}}$ can be computed by multiplying the subpart of $D$ corresponding to $(D_{m,W_m} x_{W_m})_{m \in \mathcal{E}_{V_1}}$ (see Figure 3(a), horizontal rectangle in continuous yellow lines) with $(x_{W_m})_{m \in \mathcal{E}_{R_1}} = (x_1, x_{(1,2)})$ (vertical rectangle in continuous yellow lines). This quantity is computed on worker 1, once worker 2 has communicated $x_{(1,2)}$ to worker $k = 1$. Similarly, for worker $k = 2$, the quantity $(D_{m,W_m} x_{W_m})_{m \in \mathcal{E}_{R_2}}$ can be computed by multiplying the subpart of $D$ corresponding to $(D_{m,W_m} x_{W_m})_{m \in \mathcal{E}_{R_2}}$ (horizontal rectangle in continuous green lines) with $(x_{W_m})_{m \in \mathcal{E}_{R_2}} = (x_2, x_{(2,3)})$ (vertical rectangle in continuous green lines). This quantity is computed on worker 2, once worker 3 has communicated $x_{(2,3)}$ to worker $k = 2$.

To compute $D^* \mathbf{d}$, communications are performed after computing subparts of $D^* \mathbf{d}$. In Figure 3(b), for worker $k = 1$, the quantity $[D^* \mathbf{d}]_1$ can be computed by multiplying the subpart of $D^*$ corresponding to $(D^*_{m,1})_{m \in \mathcal{E}_1}$ (continuous-line yellow rectangle, top left) with $(d_m)_{m \in \mathcal{E}_1}$. For the computation of $[D^* \mathbf{d}]_1$, no communication with other workers is needed in this example. For worker $k = 2$, $[D^* \mathbf{d}]_2$ needs to be decomposed between parts of $D^*$ stored on worker $k = 2$, and parts of $D$ that are stored on other workers, i.e., $k = 1$ for this example. For the parts of $D^*$ stored on worker $k = 2$, the subpart of $D^*$ corresponding to $(D^*_{m,2})_{m \in \mathcal{E}_2}$ (continuous-line green rectangle) needs to be multiplied with $(d_m)_{m \in \mathcal{E}_2}$. For the parts of $D^*$ stored on worker $k = 1$, the subpart of $D^*$ corresponding to $(D^*_{m,1})_{m \in \mathcal{E}_1}$ (dotted-line green rectangle) needs to be multiplied with $(d_m)_{m \in \mathcal{E}_1}$. This second part is computed on worker $k = 1$ (using only the yellow part of the dotted-line green rectangle), and then communicated and aggregated on worker $k = 2$. For worker $k = 3$, the decomposition of the quantity $[D^* \mathbf{d}]_3$ is similar to the one taken for worker $k = 2$.

5.2 Distributed SPMD architecture

The distributed block-coordinate sampler described in the previous section can benefit from an implementation on an Single Program Multiple Data (SPMD) architecture. In contrast with a client-server configuration, all the workers execute the same task on a subset of each block of parameters and observations [15]. This enables the hypergraph structure of $D$ to be exploited to reduce the number and volume of the communications.

In practice, Theorem 4.1 and Theorem 4.4 ensure that most of the operations of Algorithm (2.8) are compatible with an SPMD architecture. As noted in Section 1, the data fidelity term can be represented by either the function $f$, $g$ of $h$ in (1.2), depending on the nature of the noise $D$ affecting the observations $y \in G$ in (1.1). The separability Assumptions 4.4-(ii) and (iii) ensure that $K$ conditionally independent blocks of observations can be formed. This leads to a partition of the observations into $y = \{y_k\}_{k \in K}$, with $y_k \in G_k$ the block of observations stored on the worker $k$. In this case, no observation needs to be exchanged between the workers. The separability of $g$ further implies that the evaluation of its proximity operator is easy to parallelize. Combining the
definition of $\phi_\alpha$ and $\psi_\beta$ (Theorem 4.4-(iv)) with the structure of $D$ finally enables Algorithm (2.8) to be reformulated using blocks of parameters, each stored on (only) one of the $K$ workers. Using an SPMD architecture for Algorithm (5.1) offers several advantages listed below.

(i) Load balancing: computation costs can be equally shared among the workers, as they all operate similar tasks on a subset of (overlapping) parameters and observations.

(ii) Parallelization flexibility: an SPMD architecture can be readily used to address (1.2) under Theorem 4.4, whereas a client-server architecture cannot (no conditionally independent blocks of variables as in [26])

(iii) Memory and computing costs per worker: each worker can be assigned a block of observations and a corresponding block of parameters. This opportunity can significantly reduce the computing and memory costs per worker.

(iv) Communications and data locality: all the observations and most of the parameters required to perform operations on a worker can be directly stored on the same worker, and do not require to be communicated (data locality). The conditions given in Section 3.2 guarantee that communication costs are limited: only a few elements need to be retrieved by each worker from a small number of connected workers.

Appendix A addresses the multi-term extension of (1.2). Note that a client-server architecture can also be used to address this case. However, it cannot take advantage of the structure of the hypergraphs underlying the linear operators $(D_i)_{1 \leq i \leq I}$. The number of workers it can accommodate is also restricted to the number of conditionally independent blocks of variables, as in [35, 26].

6 Application to supervised image deconvolution

The proposed distributed SPA Gibbs sampler is applied to a supervised image deconvolution problem corrupted by Poisson noise. The induced hypergraph structures are used to adopt an SPMD strategy. Note that another application of the proposed sampler has also been studied in [31] for an inpainting problem (i.e., with $D$ a selection matrix) corrupted by additive white Gaussian noise under a TV prior.

The application example presented in this section is associated with a distribution involving two composite terms with linear operators. The notation used below thus correspond to those introduced in Appendix A for a multi-term distribution.
6.1 Problem statement

Supervised Poisson deconvolution aims at inferring an unknown image \( \bar{x} \in \mathbb{R}^N \) from observations \( y = (y_m)_{1 \leq m \leq M} \in \mathbb{R}^M \) such that

\[
(\forall m \in \{1, \ldots, M\}) \quad y_m \sim \mathcal{P}(D_1 \bar{x}),
\]

(6.1)

where \( D_1 \in \mathbb{R}^{M \times N} \) is a discrete convolution operator derived from a kernel of size \( L = L_1 \times L_2 \ll N \), and \( \mathcal{P}(\mu) \) is a Poisson distribution with mean \( \mu \). In this context, \( \bar{N} = N \) and, for every \( n \in \{1, \ldots, N\} \), \( H_n = \mathbb{R} \). Similarly, \( \bar{M} = M \) and, for every \( m \in \{1, \ldots, M\} \), \( G_m = \mathbb{R} \).

We propose to solve this problem with a hybrid prior combining a non-negativity constraint and a TV regularization [28]. Such prior has for instance been considered in [17, 33]. The resulting posterior distribution is given by

\[
\pi(x) \propto \exp\left( -f(x) - g_1(D_1x) - g_2(D_2x) \right),
\]

(6.2)

where \( g_1: \mathbb{R}^M \to ]-\infty, +\infty[ \) and \( (z_m)_{1 \leq m \leq M} \mapsto \sum_{m=1}^M g_1.m(z_m) \) is the data-fidelity term given by

\[
(\forall m \in \{1, \ldots, M\}) \quad g_1.m(z_m) = -y_m \log(z_m) + z_m,
\]

(6.3)

due to the Poisson distribution, \( f = \iota_{[0, +\infty[} : \mathbb{R}^N \to ]-\infty, +\infty[ \) is the indicator function of the positive orthant, and \( g_2 \circ D_2 \) models the discrete isotropic TV [28]. Precisely, \( D_2: \mathbb{R}^N \to \mathbb{R}^{2 \times N} \) is the concatenation of the vertical and horizontal discrete gradients, and \( g_2: \mathbb{R}^{2 \times N} \to ]-\infty, +\infty[ \) is the \( \ell_{2,1} \) -norm

\[
(\forall z = (z_n)_{1 \leq n \leq N} \in \mathbb{R}^{2 \times N}) \quad g_2(z) = \sum_{n=1}^N g_{2,n}(z_n),
\]

(6.4)

\[
(\forall n \in \{1, \ldots, N\}) (\forall z_n \in \mathbb{R}^2) \quad g_{2,n}(z_n) = \kappa \|z_n\|_2, \quad \text{where } \kappa > 0.
\]

As a first step towards a distributed implementation, the AXDA approach (see Appendix A for details) is used to approximate \( \pi(x) \) by

\[
\pi_{\alpha,\beta}(x, (z_i, u_i)_{1 \leq i \leq 2}) \propto \exp\left( -f(x) - \sum_{i=1}^2 (g_i(z_i) + \phi_{i,\alpha}(D_1x, z_i - u_i) + \psi_{i,\beta}(u_i)) \right),
\]

(6.5)

where \((\alpha_1, \alpha_2, \beta_1, \beta_2) \in [0, +\infty[^4 \) and the functions \((\phi_{i,\alpha}, \psi_{i,\beta})_{1 \leq i \leq 2} \) are defined in (2.3).

The directed acyclic graph reported in Figure 4 summarizes the structure of the approximate posterior distribution (6.5), highlighting conditional dependencies between variables. Note that \((z_1, u_1)\) and \((z_2, u_2)\) are the only conditionally independent blocks of variables.
Figure 4: Directed acyclic graph describing the factorization of the approximate posterior distribution (6.5). Circed variables correspond to random variables. Other variables are fixed a priori.

6.2 Proposed SPMD implementation

The proposed approach is applicable when a single operator is involved in the model (as in Section 5), but also when multiple operators are involved (see Appendix A), as in the present example. Note that a client-server architecture could also be considered. However, it would drastically increase the communication costs, as full-size variables would need to be duplicated on all workers or exchanged (see Section 5.2 for more details).

We propose to use the proposed distributed SPA Gibbs sampler (A.6) with $I = 2$ linear operators. In addition, for every $t \in \{0, \ldots, T\}$ and $k \in \{1, \ldots, K\}$, we use PSGLA transitions to compute $(z_{1,k}^{(t+1)}, z_{2,k}^{(t+1)})$, while $(u_{1,k}^{(t+1)}, u_{2,k}^{(t+1)})$ are drawn from their conditional distribution. More precisely, we have, for $i \in \{1, 2\}$,

$$z_{i,k}^{(t+1)} = \text{prox}_{g_{i,k}} \left( z_{i,k}^{(t)} - \eta_i \alpha_i^{-2} (z_{i,k}^{(t)} - v_{i,k}^{(t+1)} + u_{i,k}^{(t)}) + \sqrt{2\eta_i} \xi_{i,k}^{(t)} \right) ,$$

(6.6)

where $g_{i,k} = \sum_{m \in E_{i,k}} g_{i,m}$, $\eta_i \in [0, \alpha_i^{-2}]$, and $\xi_{i,k}^{(t)} \sim \mathcal{N}(0, I)$. The proximity operators involved in the resulting algorithm can be found, e.g., in [17, 19]. The distributed implementation of $(D_1)_{1 \leq i \leq 2}$ and $(D_1^\ast)_{1 \leq i \leq 2}$ is detailed below, using a 2D Cartesian grid of $K$ workers.

Distributed implementation of $D_1$ Figure 5 illustrates the preliminary communications required by the distributed implementation of $D_1$. Each worker $k \in \{1, \ldots, K\}$ needs to collect $(x_{1,(k,k')}^{(k-k')})_{k' \in \mathcal{R}_k}$, as illustrated in Figure 5(a). A first communication step occurs along the horizontal axis. Each worker $k$ sends vertices from its left-hand-side border (of width $L_2$, the horizontal width of the blur kernel) to its neighbour on the left (Figure 5(a), dark blue areas), and receives vertices from the neighbour on its right (Figure 5(a), light blue areas). Once the first step is complete, a second communication step occurs along the vertical axis with the top and bottom neighbours of the worker $k$ (Figure 5(a), dark and light red areas, of width $L_1$). The local operator $(D_{1,m})_{m \in E_{1,k}}$ is equivalent to considering a convolution matrix and a selection operator. The latter ensures that the correct boundaries are considered (only the convolution outputs which have not interacted with the boundaries of $(x_{1,m}^{(m)})_{m \in E_{1,k}}$ are valid).
Figure 5: Communication patterns involved in the distributed implementation of (a) $D_1$ and (b) $D'_1$. Each worker $k$ is required to communicate with two contiguous workers along each direction successively. Colored arrows indicate whether vertices (a) or hyperedge weights (b) are received from or sent to a nearby worker. Vertices (a) and hyperedge weights (b) sent and received along the horizontal and vertical communication steps are highlighted in blue and red, respectively. (b) The circled “plus” symbols represent contributions aggregated with the corresponding hyperedge weights of the reception worker. These operations correspond to the second term in (5.14).
Distributed implementation of $D_1^*$ The distributed implementation of $D_1^*$ is similar to that of $D_1$. A first communication step occurs along the horizontal axis. Each worker $k$ sends hyperedge weights from its right-hand-side border (of width $L_2$) to its neighbour on the right (Figure 5(b), dark blue areas), and aggregates weights from the neighbour on its left (Figure 5(b), light blue areas). A similar communication step then occurs along the vertical axis with the bottom and top neighbours of the worker $k$ (Figure 5(b), dark and light red areas, of width $L_1$). A convolution is then applied on each worker to all the weights locally available, followed by a selection operator to ensure correct boundaries have been used.

Distributed implementation of $D_2$ The distributed implementation of $D_2$ requires the same communication pattern as $D_1$, successively exchanging messages with width 1 along the horizontal and vertical directions. The operator $(D_{2,m},\mathcal{V}_{2,m})_{m \in \mathcal{E}_{2,n_k}}$ corresponds to a local discrete gradient operator, using the boundaries retrieved during the communication step. Note that, for $k \in \{1, \ldots, K\}$ and $k' \in \mathcal{R}_k$, the set of vertices $\mathcal{V}_{2,(k,k')}$ to be communicated by the worker $k$ to $k'$ is such that $\mathcal{V}_{2,(k,k')} \subset \mathcal{V}_{1,(k,k')}$. The total number of elements to be communicated in the algorithm is thus reduced, as vertices required by the distributed implementation of $D_2$ already need to be communicated for $D_1$.

Distributed implementation of $D_2^*$ The distributed implementation of $D_2^*$ requires the same communication pattern as $D_1^*$, successively exchanging messages with width 1 along the horizontal and vertical directions. A local discrete adjoint gradient operator is then applied on each worker $k$ to all the hyperedge weights available locally.

6.3 Experiments

The proposed approach is evaluated in terms of estimation quality and scalability on the deconvolution problem of Section 6.1. Results are compared with those of the reference serial SPA Gibbs sampler from [33].

6.3.1 Simulation setting

All the experiments have been conducted on a single computer equipped with two 2.1 GHz, 18-core, Intel Xeon E5-2695 v4 series processors (36 CPU cores in total). In this setting, a worker corresponds to a process running on one CPU core. The proposed distributed sampler has been implemented in Python using the mpi4py library [14]. A short code to reproduce part of the experiments is available at https://gitlab.cristal.univ-lille.fr/pthouven/dspa and will be completed upon acceptance of the paper.
The proposed approach is compared with the SPA Gibbs sampler proposed in [33]. The latter relies on a different splitting strategy compared to Section 6.1 using $I = 3$ operators (see [33] for further details). In practice, MYULA transition kernels are leveraged to sample from conditional distributions involving non-smooth potential functions. This choice of transition kernel and splitting strategy requires the proximity operator of the TV norm to be evaluated at each iteration of the sampler, using a primal-dual algorithm [6].

Performance assessment is conducted in terms of runtime per iteration and quality of both the minimum mean square error (MMSE) and maximum a posteriori (MAP) estimators. The estimators are denoted $x_{\text{MMSE}}$ and $x_{\text{MAP}}$, respectively. Reconstruction quality is quantified through the structural similarity index (SSIM) [37] and the signal-to-noise ratio (SNR) expressed in dB. The associated 95% confidence intervals are also reported.

The sampler [33] has been systematically applied with $(\alpha_i^2, \beta_i^2)_{1 \leq i \leq 3, \kappa} = 1_7$, where $1_Q \in \mathbb{R}^Q$ is a vector with entries all equal to 1. The proposed approach uses $((\alpha_i^2, \beta_i^2)_{1 \leq i \leq 2, \kappa}) = 1_5$. For both algorithms, $N_{\text{MC}} = 5 \times 10^3$ samples have been generated to form $x_{\text{MMSE}}$ and $x_{\text{MAP}}$ and the 95% credibility intervals, discarding $N_{\text{bi}} = 2 \times 10^3$ burn-in samples. Two series of experiments have been considered to assess the estimation quality and scalability.

6.3.2 Experiment results

Estimation quality Using $K = 1$ worker, the proposed approach is compared with [33]. Ground truth images with different maximum intensity levels $\tau_{\text{max}} = \max_{1 \leq n \leq N} \tau_n$ have been considered. The values $\tau_{\text{max}} \in \{20, 30\}$ have been adopted for the following datasets:

(i) house image ($N = 256^2$), using a normalized Gaussian kernel of size $L \in \{3^2, 7^2\}$;
(ii) peppers image ($N = 512^2$) using a normalized Gaussian kernel of size $L \in \{7^2, 15^2\}$.

The results reported in Table 3 show that the estimators formed with the proposed approach have higher quality metrics compared to the method proposed in [33]. In addition, the computing time required by the proposed sampler is between 1.5 and 2 times smaller than [33]. This discrepancy comes from the difference in the splitting strategies considered by the two methods. In particular, the splitting approach proposed in [33] requires the evaluation of the proximal operator of the TV norm, obtained as the output of an iterative optimization algorithm. The difference in splitting can also affect the quality of the resulting AXDA approximation, as can be observed by the difference in quality of the estimator produced. The MMSE estimator reported in Figure 6 for [33] appears much smoother compared to the proposed algorithm for the same value of the regularization parameter $\kappa$. Note that the uncertainty level of the proposed approach is slightly lower and appears more diffuse than [33].
| Dataset   | Algo.       | SNR($x_{\text{MMSE}}$) | SNR($x_{\text{MAP}}$) | SSIM($x_{\text{MMSE}}$) | SSIM($x_{\text{MAP}}$) | Time per iter. $\times 10^{-2}$ s | Runtime $\times 10^2$ s |
|-----------|-------------|-------------------------|------------------------|--------------------------|--------------------------|----------------------------------|------------------------|
| House     | Proposed    | 18.37                   | 15.08                  | 0.60                     | 0.20                     | 5.59 (0.12)                      | 1.68                   |
| (L = 3$^2$) | Proposed    | 18.00                   | 14.92                  | 0.60                     | 0.20                     | 8.52 (0.22)                      | 2.56                   |
| Peppers   | Proposed    | 18.98                   | 15.08                  | 0.59                     | 0.24                     | 4.88 (0.13)                      | 1.47                   |
| (L = 7$^2$) | Proposed    | 20.52                   | 16.07                  | 0.66                     | 0.25                     | 16.71 (0.38)                     | 5.01                   |
| (L = 15$^2$) | Proposed    | 18.90                   | 15.07                  | 0.66                     | 0.25                     | 36.68 (0.33)                     | 11.01                  |

### Table 3: Comparison between [33] and the proposed approach with $K = 1$. Datasets have been generated from ground truth images with different maximum intensity $\tau_{\text{max}} \in \{20, 30\}$ and convolution kernel sizes $L \in \{3, 7, 15\}$. Results are reported in terms of estimation quality, runtime per iteration and total runtime.

![Observations and ground truth](image1)

![MMSE estimator and 95% credible interval](image2)

![MMSE estimator and 95% credible interval](image3)

Figure 6: Estimators and 95% credibility intervals for peppers, with $\tau_{\text{max}} = 30$ and $L = 15^2$.  

28
Strong scaling experiment  For this experiment, the behaviour of the proposed method is investigated with a varying number of workers $K \in \{1, 2, 4, 8, 16, 32\}$, using the maximum intensity level $\tau_{\text{max}} = 30$ for the peppers dataset.

Table 4 and Table 5 show the speedup for the proposed approach when $L = 7^2$ and $L = 15^2$, respectively. In Table 5, a close to ideal speedup (i.e., close to the number of workers $K$) is observed as $L = 15^2$. Note that speedup factors larger than the number of cores may occur, depending on the cache state of the machine at the time the experiments have been run. In comparison, the ideal runtime per iteration (i.e., neglecting communication time) for [33] for a client-server architecture with $K = 4$ cores is $1.48 \times 10^{-1}$ s when $L = 7^2$ ($1.04 \times 10^{-1}$ s for TV-related terms), and $3.41 \times 10^{-1}$ s when $L = 15^2$ ($1.02 \times 10^{-1}$ s for TV-related terms). In both cases, the time required to update the other splitting variables is almost 10 times lower. For this experiment, a client server approach thus leads to a limited runtime performance, given the heterogeneity in the complexity of the tasks assigned to workers. All these results illustrate the ability of the proposed sampler to provide estimators at a fraction of the runtime of the serial implementation by using an increasing number of cores $K$.

Weak scaling experiment  For this experiment, the behaviour of the proposed method is investigated when both the size of the problem and the number of workers considered simultaneously increase. A fixed problem size per worker is considered, using $K \in \{1, 4, 16\}$. Datasets derived from upsampled versions of the house image are considered, with a maximum intensity level $\tau_{\text{max}} = 30$, using $(N, L) = \{(256^2, 3^2), (512^2, 7^2), (1022^2, 11^2)\}$. These numbers ensure that both the problem size $M$ and the associated number of workers $K$ evolve in the same proportions over the configurations tested.

Table 6 shows scaled speedup factors (that is, normalized by the factor of increase for $K$ and $M$) close to the number of cores used. Differences from a linear scaling may result from fixed communication costs, representing a larger cost per iteration as $L$ increases. Overall, the results illustrate the runtime stability of the approach for a fixed problem size per worker. This experiment efficiently processes a 1 million pixel image in about 2 minutes to obtain an estimator with the associated credibility intervals.

7 Conclusion

In this paper, a distributed block-coordinate SPA Gibbs sampler has been introduced to efficiently solve large scale imaging inverse problems. The approach leverages the approximate data augmentation scheme AXDA [35, 26] to efficiently handle composite functions involving linear operators. A block-coordinate approach is adopted to split and distribute all the variables over multiple workers. The proposed method exploits the hypergraph structure of the linear operators to design a versatile distributed block-coordinate sampler.
| $K$ | SNR($x_{MMSE}$) | SNR($x_{MAP}$) | SSIM($x_{MMSE}$) | SSIM($x_{MAP}$) | Time per iter. ($\times 10^{-2}$ s) | Speedup | Runtime ($\times 10^2$ s) |
|-----|----------------|----------------|------------------|----------------|--------------------------------------|---------|--------------------------|
| 1   | 20.71          | 17.96          | 0.71             | 0.40           | 10.26 (1.22)                         | 1.00    | 3.08                     |
| 2   | 20.70          | 17.94          | 0.71             | 0.40           | 5.33 (0.10)                          | 1.93    | 1.60                     |
| 4   | 20.72          | 17.93          | 0.71             | 0.40           | 4.35 (0.12)                          | 2.36    | 1.30                     |
| 8   | 20.72          | 17.95          | 0.71             | 0.40           | 2.51 (0.20)                          | 4.08    | 0.75                     |
| 16  | 20.73          | 17.97          | 0.71             | 0.41           | 1.23 (0.03)                          | 8.32    | 0.37                     |
| 32  | 20.71          | 17.91          | 0.71             | 0.40           | 0.60 (0.06)                          | 17.04   | 0.18                     |

Table 4: Results of the strong scaling experiment using a dataset with $\pi_{\text{max}} = 30$ and kernel size $L = 7^2$. Performance is reported in terms of estimation quality, time per iteration and speedup.

| $K$ | SNR($x_{MMSE}$) | SNR($x_{MAP}$) | SSIM($x_{MMSE}$) | SSIM($x_{MAP}$) | Time per iter. ($\times 10^{-2}$ s) | Speedup | Runtime ($\times 10^2$ s) |
|-----|----------------|----------------|------------------|----------------|--------------------------------------|---------|--------------------------|
| 1   | 20.74          | 17.98          | 0.71             | 0.41           | 23.22 (0.34)                         | 1.00    | 6.97                     |
| 2   | 20.72          | 17.94          | 0.71             | 0.40           | 12.42 (0.11)                         | 1.87    | 3.73                     |
| 4   | 20.74          | 17.98          | 0.71             | 0.41           | 3.22 (0.10)                          | 7.22    | 0.97                     |
| 8   | 20.74          | 17.97          | 0.71             | 0.41           | 1.91 (0.06)                          | 12.17   | 0.57                     |
| 16  | 20.72          | 17.95          | 0.71             | 0.40           | 1.31 (0.11)                          | 17.73   | 0.39                     |
| 32  | 20.73          | 17.97          | 0.71             | 0.40           | 0.70 (0.07)                          | 33.35   | 0.21                     |

Table 5: Results of the strong scaling experiment using a dataset with $\pi_{\text{max}} = 30$ and kernel size $L = 15^2$. Performance is reported in terms of estimation quality, time per iteration and speedup.

| $(N, L, K)$ | SNR($x_{MMSE}$) | SNR($x_{MAP}$) | SSIM($x_{MMSE}$) | SSIM($x_{MAP}$) | Time per iter. ($\times 10^{-2}$ s) | Sc. speedup | Runtime ($\times 10^2$ s) |
|-------------|----------------|----------------|------------------|----------------|--------------------------------------|-------------|--------------------------|
| $(256^2, 3^2, 1)$ | 20.18          | 17.84          | 0.66             | 0.34           | 2.03 (0.05)                          | 1.00        | 0.61                     |
| $(512^2, 7^2, 4)$ | 23.86          | 19.56          | 0.74             | 0.34           | 2.13 (0.08)                          | 3.81        | 0.64                     |
| $(1022^2, 11^2, 16)$ | 27.55          | 20.65          | 0.81             | 0.33           | 3.56 (0.18)                          | 9.13        | 1.07                     |

Table 6: Results of the weak scaling experiment using a dataset with $\pi_{\text{max}} = 30$. The reconstruction quality of the estimators is reported with the time per iteration, the scaled speedup and the runtime.
Experiments on a supervised image deblurring problem show that the proposed approach forms reliable estimates with quantified uncertainty in a significantly reduced amount of time, compared to a state-of-the-art non-distributed version of the sampler [33]. In particular, the proposed sampler is shown to provide estimators at a fraction of the runtime of the serial implementation by using an increasing number of workers $K$. Processing a 1 million pixel image using our current Python implementation takes less than 2 minutes to obtain a good restoration with associated credibility intervals.

Note that the proposed distributed block-coordinate SPA Gibbs sampler is directly applicable to a much wider class of applications than the restoration problems addressed in this work, such as image inpainting, super-resolution or reconstruction. Future works include the development of an asynchronous version of the proposed approach, to further speed up the inference process, while maintaining the convergence of the Markov chain.

A SPA Gibbs sampler for multiple composite terms

A.1 AXDA for multiple composite terms

Following [35], AXDA can be generalized to handle multiple composite terms. We will use similar ideas to generalize the proposed approach.

Let $\mathcal{M} = \sum_{i=1}^{I} M_i$, and for every $i \in \{1, \ldots, I\}$, $G_i = \mathbb{R}^{\mathcal{M}_i}$. We propose to generalize model (1.2) by considering a posterior distribution of the form of

$$\pi(x) \propto \exp \left( -h(x) - f(x) - \sum_{i=1}^{I} g_i(D_i x) \right), \quad (A.1)$$

where, for every $i \in \{1, \ldots, I\}$, $D_i : \mathcal{H} \rightarrow G_i$ and $g_i : G_i \rightarrow ]-\infty, +\infty[$. Similarly to (1.2)-(1.4), (A.1) can be approximated by

$$\pi(\alpha, \beta)(x, (z_i, u_i)_{1 \leq i \leq I}) \propto \exp \left( -h(x) - f(x) - \sum_{i=1}^{I} \left( g_i(z_i) + \phi_{i, \alpha_i}(D_i x, z_i - u_i) + \psi_{i, \beta_i}(u_i) \right) \right), \quad (A.2)$$

where $(z_i, u_i)_{1 \leq i \leq I}$ are auxiliary variables and, for $i \in \{1, \ldots, I\}$, $\phi_{i, \alpha_i} : \mathcal{G}_i \times \mathcal{G}_i \rightarrow ]-\infty, +\infty[,$ $\psi_{i, \beta_i} : \mathcal{G}_i \rightarrow ]-\infty, +\infty[.$ The parameters $(\alpha, \beta) = (\alpha_i, \beta_i)_{1 \leq i \leq I}$ control the discrepancy between $(D_i x)_{1 \leq i \leq I}$ and $(z_i)_{1 \leq i \leq I}$. In [35], the authors show that Proposition 2.1 holds in this context, under the same conditions on $\phi_{i, \alpha_i}$ and $\psi_{i, \beta_i}$ as (2.1)-(2.2).

Using an approach similar the one described in Section 2.2, we can design a PSGLA within
Given by operator $D_i$ user

| Notation | Definition | |
|---|---|---|
| $e_{i,m} = \{e_{i,m}\}_{i\in M_i}$ | Hyperedges of $H_i$ | ✓ |
| $e_{i,m} \subset \{1, \ldots, N\}$ | Vertex indices in hyperedge $m$ of $H_i$ | ✓ |
| $k_{i,m} \in \{1, \ldots, K\}$ | Worker associated with $m$-th hyperedge $e_{i,m}$ (chosen by the user). $k_{i,m}$ must satisfy $e_{i,m} \cap \cup_{k_{i,m}} \neq \emptyset$ | ✓ |
| $W_{i,m} \subset \{1, \ldots, K\} \setminus \{k_{i,m}\}$ | Set of all workers but $k_{i,m}$, containing vertices from $e_{i,m}$ | ✓ |
| $\bar{W}_{i,m} \subset \{1, \ldots, K\}$ | $\bar{W}_{i,m} = k_{i,m} \cup W_{i,m}$ | ✓ |
| $\forall_{i,k,k'} \subset \mathcal{V}_k'$ | Indeces of vertices sent from worker $k'$ to worker $k$ | ✓ |
| $\forall_{W_{i,m}} \subset \{1, \ldots, N\} \setminus \forall_{k_{i,m}}$ | $\forall_{W_{i,m}} = \bigcup_{k' \in W_{i,m}} \forall_{(k_{i,m}, k')}$ the set of vertex indices that will be communicated to worker $k_{i,m}$ from all workers $k' \in W_{i,m}$ | ✓ |
| $\forall_{\pi_{i,m}} \subset \{1, \ldots, N\}$ | $\forall_{\pi_{i,m}} = \forall_{k_{i,m}} \cup \forall_{W_{i,m}}$ the set of vertex indices necessary to perform computations associated with $k_{i,m}$ | ✓ |
| $\mathcal{E}_{i,k} \subset \{1, \ldots, M_i\}$ | Indices of hyperedges only containing vertices stored on worker $k$ | ✓ |
| $\mathcal{E}_{i,(k,k')} \subset \{1, \ldots, M_i\}$ | Indices of hyperedges containing vertices sent from worker $k'$ to worker $k$ | ✓ |
| $\mathcal{E}_{i,R_k} \subset \{1, \ldots, M_i\}$ | $\mathcal{E}_{i,R_k} = \bigcup_{k' \in R_k} \mathcal{E}_{i,(k,k')}$ set of all hyperedges containing vertices communicated to worker $k$ | ✓ |
| $\mathcal{E}_{i,k} \subset \{1, \ldots, M_i\}$ | $\mathcal{E}_{i,k} = \mathcal{E}_{i,k} \cup \mathcal{E}_{i,R_k}$, such that $(\mathcal{E}_{i,k})_{i \in K}$ is a partition of $\{1, \ldots, M_i\}$ | ✓ |

Table 7: Summary of the set notation used to define the hypergraph structure associated with the operators $D_i$, for $i \in \{1, \ldots, I\}$. These notation generalize those provided in Table 1.

Gibbs sampler to approximate (A.2), and generalizing Algorithm (2.8) as follows

\[
\begin{align*}
\text{for } t = 0, 1, \ldots, T \\
x^{(t+1)} &= \text{prox}_{\frac{1}{\gamma}h}(x^{(t)}) - \gamma \sum_{i=1}^{I} D_i^* \left( \nabla \phi_{i,\alpha_i}(\cdot, z^{(t)}_i - u^{(t)}_i)(v^{(t)}_i) \right) + \sqrt{2\gamma}w^{(t)}_i, \\
\text{for } i = 1, \ldots, I \\
v^{(t+1)}_i &\sim D_i^* x^{(t+1)}, \\
z^{(t+1)}_i &\sim K_{z_i,\alpha_i}(v^{(t+1)}_i, u^{(t)}_i, z^{(t)}_i), \\
u^{(t+1)}_i &\sim N \left( \frac{\beta^2}{\alpha^2 + \beta^2} (z^{(t+1)}_i - v^{(t+1)}_i), \frac{\alpha^2 + \beta^2}{\alpha^2} I \right), \\
\end{align*}
\]

(A.3)

where $(w^{(t)})_{0 \leq t \leq T}$ is a sequence of independent and identically distributed (i.i.d) standard Gaussian random variables in $\mathbb{H}$, $\gamma \in [0, \lambda^{-1}]$, $\lambda = \lambda_h + \|\sum_{i=1}^{I} D_i/\alpha_i\|_2$, and, for every $i \in \{1, \ldots, I\}$, $K_{z_i,\alpha_i}$ is a transition kernel with invariant distribution

\[
(\forall i \in \{1, \ldots, I\}) \quad \pi_{i,\alpha_i}(z_i \mid v_i, u_i) \propto \exp \left( -g_i(z_i) - \phi_{i,\alpha_i}(v_i, z_i - u_i) \right),
\]

(A.4)

for $v_i = D_i x$.

32
Definition

Table 8: Notation used for the variables associated with the hypergraph induced by the operators $D_i$, for $i \in \{1, \ldots, I\}$. These notation generalize those provided in Table 2.

### A.2 Distributed multi-term SPA Gibbs sampler

This section introduces a distributed version of Algorithm (A.3), using the same approach as in Section 4 and 5. We thus consider the distribution (A.1), with its AXDA approximation (A.2).

For every $i \in \{1, \ldots, I\}$, let $G_i = \mathbb{R}^{M_i}$ such that $G_i = G_{i,1} \times \ldots \times G_{i,M_i}$, where for every $m \in \{1, \ldots, M_i\}$, $G_{i,m} = \mathbb{R}^{M_{i,m}}$, with $M_i = \sum_{m=1}^{M_i} M_{i,m}$. An element of $G_i$ is denoted by $u_i = (u_{i,m})_{1 \leq m \leq M_i}$, where, for every $m \in \{1, \ldots, M_i\}$, $u_{i,m} \in G_{i,m}$.

For every $i \in \{1, \ldots, I\}$, the linear operator $D_i = (D_{i,m,n})_{1 \leq m \leq M_i, 1 \leq n \leq N}$ defines a hypergraph structure $H_i$ as described in Section 4. We thus consider $I$ hypergraphs, distributed over the same $K$ workers. As in Section 4, the choice of the distribution is made by the user, depending on the shape of the hypergraphs. The associated notation given in Table 7 and Table 8 generalize those introduced in Section 4.2. For every $k \in \{1, \ldots, K\}$, let $G_{i,k} = \times_{m \in \mathbb{E}_{i,k}} G_{i,m}$ such that $G_i = \times_{1 \leq k \leq K} G_{i,k}$.

We assume that, for every $i \in \{1, \ldots, I\}$, the functions $g_i$, $\phi_{i,\alpha}$, and $\psi_{i,\beta}$ satisfy the same separability assumptions as the functions $g$, $\phi_{\alpha}$, $\psi_{\beta}$ given in Theorem 4.4. Then, for every $i \in \{1, \ldots, I\}$, there exists a permutation operator $\vartheta_i : G_i \rightarrow G_i$ such that, for every $x \in H$,

$$
D_i x = \left( (D_{i,m,n})_{1 \leq n \leq N} x \right)_{1 \leq m \leq M_i} = \vartheta_i \left( \left( D_{i,m,k} x_{i,k} \right)_{m \in \mathbb{E}_{i,k}} \right)_{1 \leq k \leq K},
$$

(A.5)
and, for every $u_i \in G_i$, we have

$$g_i(u_i) = \sum_{k=1}^{K} \left( \sum_{m \in E_{i,k}} g_{i,m}(u_{i,m}) \right),$$

$$\phi_{i,\alpha_i}(D_i x, u_i) = \sum_{k=1}^{K} \left( \sum_{m \in E_{i,k}} \phi_{i,m,\alpha_i}(D_{i,m,k} x_{i,k}, u_{i,m}) + \sum_{m \in E_{i,R,k}} \phi_{i,m,\alpha_i}(D_{\psi_{i,m,\alpha_i}} x_{\psi_{i,m,\alpha_i}}, u_{i,m}) \right),$$

$$\psi_{i,\beta_i}(u) = \sum_{k=1}^{K} \left( \sum_{m \in E_{i,k}} \psi_{i,m,\beta_i}(u_{i,m}) \right).$$

Using these notation, Theorem 5.1 can be generalized to a multi-term setting as follows.

**Proposition A.1** Assume that, for every $i \in \{1, \ldots, I\}$, each operator $D_i$ is split over workers $\{1, \ldots, K\}$ such that, for every $k \in \{1, \ldots, K\}$, $(D_{i,m,k})_{m \in \mathbb{E}_{i,k}}$ is stored on worker $k$. Let, for every $k \in \{1, \ldots, K\}$, $x_k^{(0)} \in H_k$, $z_{i,k}^{(0)} \in G_{i,k}$, and $u_k^{(0)} \in G_{i,k}$. Let $(x^{(t)})_{1 \leq t \leq T} \text{ and } (z_{i,t}^{(t)}, u_{i,t}^{(t)})_{1 \leq i \leq I, 1 \leq t \leq T}$
be samples generated by the following algorithm:

\[
\text{for } k = 1, \ldots, K \\
\quad \text{for } k' \in S_k \\
\quad \quad \text{Send } (x_{i,k}^{(0)})_{n \in \cup_{i=1}^{I} V_{i,(k',k)}} \text{ to worker } k',
\]

\[
\text{for } k' \in R_k \\
\quad \text{Receive } (x_{i,k}^{(0)})_{n \in \cup_{i=1}^{I} V_{i,(k,k')}} \text{ from worker } k',
\]

\[
\text{for } i = 1, \ldots, I \\
\quad v_{i,k}^{(0)} = \left( D_{i,m,k} x_{i,k}^{(0)} \right)_{m \in E_i,k},
\]

\[
\text{for } t = 0, 1, \ldots, T \\
\text{for } k = 1, \ldots, K \\
\quad \text{for } i = 1, \ldots, I \\
\quad \quad (d_{i,m,k}^{(t)})_{m \in E_i,k} = \left( \phi_{i,m,\alpha_i}(\cdot, z_{i,m}^{(t)} - u_{i,m}^{(t)})(v_{i,m}^{(t)}) \right)_{m \in E_i,k},
\]

\[
\text{for } k' \in S_k \\
\quad \text{Receive } d_{i,m,k}^{(t)} \text{ from worker } k',
\]

\[
\nabla_k^{(t)} = \sum_{i=1}^{I} \sum_{m \in E_i,k} D_{i,m,k} d_{i,m}^{(t)} + \sum_{k' \in S_k} d_{i,m,k}^{(t)},
\]

\[
x_{k}^{(t+1)} = \text{prox}_{\gamma f_k} \left( x_{k}^{(t)} - \gamma \nabla h_k(x_{k}^{(t)}) - \gamma \nabla k^{(t)} + \sqrt{2\gamma} w_{k}^{(t)} \right),
\]

\[
\text{for } k' \in S_k \\
\quad \text{Send } (x_{i,k}^{(t+1)})_{n \in \cup_{i=1}^{I} V_{i,(k',k)}} \text{ to worker } k',
\]

\[
\text{for } k' \in R_k \\
\quad \text{Receive } (x_{i,k}^{(t+1)})_{n \in \cup_{i=1}^{I} V_{i,(k,k')}} \text{ from worker } k',
\]

\[
\text{for } i = 1, \ldots, I \\
\quad v_{i,k}^{(t+1)} = \left( D_{i,m,k} x_{i,k}^{(t+1)} \right)_{m \in E_i,k},
\]

\[
\text{for } i = 1, \ldots, I \\
\quad z_{i,k}^{(t+1)} \sim K_{z_{i,k},\alpha_i} \left( v_{i,k}^{(t+1)} : z_{i,k}^{(t)} : u_{i,k}^{(t)} \right),
\]

\[
\text{for } i = 1, \ldots, I \\
\quad v_{i,k}^{(t+1)} \sim \mathcal{N} \left( \frac{\beta_1^2}{\alpha_1^2 + \beta_1^2} (z_{i,k}^{(t+1)} - v_{i,k}^{(t+1)}) : \frac{\alpha_1^2 + \beta_1^2}{\alpha_1^2 + \beta_1^2} I \right),
\]

where \( \gamma \in \left(0, 1\right) \) and \( \frac{1}{2} (\lambda_k + \sum_{i=1}^{T} \|D_i\|^2 / \alpha_i^2)^{-1} \) for every \( k \in \{1, \ldots, K\} \), \((w_{k}^{(t)})_{1 \leq t \leq T}\) is a sequence of i.i.d. standard Gaussian random variables in \( H_{k} \). In addition, for every \( i \in \{1, \ldots, I\} \), \( K_{z_{i,k},\alpha_i} \) is a
transition kernel with invariant distribution

\[
\pi_{i,k,\alpha}(z_{i,k} | v_{i,k}, u_{i,k}) \propto \exp \left( - \sum_{m \in E_{i,k}} (g_{i,m}(z_{i,m}) + \phi_{i,m,\alpha}(D_{i,m,k}x_k, z_{i,m} - u_{i,m})) \right. \\
\left. - \sum_{m \in E_{i,R_k}} (g_{i,m}(z_{i,m}) + \phi_{i,m,\alpha}(D_{i,m,R_k}x_k, z_{i,m} - u_{i,m})) \right),
\]

where, for every \( x \in \mathcal{H} \),

\[
v_{i,k} = (v_{i,m})_{m \in E_{i,k}} = (D_{i,m,k}x_k)_{m \in E_{i,k}} = (D_{i,m,R_k}x_k)_{m \in E_{i,R_k}}.
\]

Then Algorithm (A.6) is equivalent to Algorithm (A.3).

Proof. The proof of Proposition A.1 is similar to the one of Proposition 5.1. □

References

[1] H. H. Bauschke and P. L. Combettes, Convex analysis and monotone operator theory in Hilbert spaces, Springer, New York, NY, 2017.

[2] A. Beck and M. Teboulle, A fast iterative shrinkage-thresholding algorithm for linear inverse problems, SIAM J. Imaging Sciences, 2 (2009), pp. 183–202.

[3] J. Birdi, A. Repetti, and Y. Wiaux., Sparse interferometric Stokes imaging under polarization constraint (Polarized SARA), Monthly Notices of the Royal Astronomical Society, 478 (2018), pp. 4442–4463.

[4] J. Bolte, S. Sabach, and M. Teboulle, Proximal alternating linearized minimization for nonconvex and nonsmooth problems, Mathematical Programming, 162 (2014), pp. 459–494.

[5] X. Cai, M. Pereyra, and J. D. McEwen, Uncertainty quantification for radio interferometric imaging – I. Proximal MCMC methods, Monthly Notices of the Royal Astronomical Society, 480 (2018), pp. 4154–4169.

[6] A. Chambolle and T. Pock, A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging, J. Math. Imaging Vision, 40 (2011), pp. 120–145.

[7] E. Chouzenoux, J.-C. Pesquet, and A. Repetti, Variable metric forward–backward algorithm for minimizing the sum of a differentiable function and a convex function, J. Optim. Theory Appl., 162 (2014), pp. 107–132.

[8] E. Chouzenoux, J.-C. Pesquet, and A. Repetti, A block coordinate variable metric forward–backward algorithm, J. Global Optim., 66 (2016), pp. 457–485.

[9] P. L. Combettes and J. Eckstein, Asynchronous block-iterative primal-dual decomposition methods for monotone inclusions, Math. Program., 168 (2018), pp. 645–672.
[10] P. L. Combettes and J.-C. Pesquet, *Proximal splitting methods in signal processing*, vol. 49, Springer, May 2011, pp. 185–212.

[11] P. L. Combettes and J.-C. Pesquet, "Stochastic quasi-fejér block-coordinate fixed point iterations with random sweeping," SIAM Journal on Optimization, 25 (2015), pp. 1221–1248.

[12] P. L. Combettes and B. C. Vî, "Variable metric forward-backward splitting with applications to monotone inclusions in duality," Optimization, 63 (2014), pp. 1289–1318.

[13] L. Condat, *A primal-dual splitting method for convex optimization involving Lipschitzian, proximale and linear composite terms*, J. Opt. Soc. America, 158 (2013), pp. 460–479.

[14] L. Dalcin and Y.-L. L. Fang, *mpi4py: Status update after 12 years of development*, IEEE Comput. Sci. Eng., 23 (2021), pp. 47–54.

[15] 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., Berlin, Heidelberg, 2001, pp. 1–1.

[16] A. Durmus, E. Moulines, and M. Pereyra, *Efficient Bayesian Computation by Proximal Markov Chain Monte Carlo: When Langevin Meets Moreau*, SIAM J. Imaging Sci., 11 (2018), pp. 473–506.

[17] M. A. T. Figueiredo and J. M. Bioucas-Dias, *Restoration of Poissonian Images Using Alternating Direction Optimization*, IEEE Trans. Image Process., 19 (2010), pp. 3133–3145.

[18] G. L. Jones, G. O. Roberts, and J. S. Rosenthal, *Convergence of conditional Metropolis-Hastings samplers*, Advances in Applied Probability, 46 (2014), pp. 422–445.

[19] 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., 32 (2015), pp. 31–54.

[20] L. Lima Pilla, *Optimal Task Assignment for Heterogeneous Federated Learning Devices*, in 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), May 2021, pp. 661–670, https://doi.org/10.1109/IPDPS49936.2021.00074.

[21] Z. Luo and P. Tseng, *On the convergence of the coordinate descent method for convex differentiable minimization*, J. Optim. Theory Appl., 72 (1992), pp. 7–35.

[22] S. Mallat, *A Wavelet Tour of Signal Processing*, Academic Press, Boston, 3rd edition ed., 2009.

[23] Y. E. Nesterov, *A method for solving the convex programming problem with convergence rate O(1/k²)*, Dokl. Akad. Nauk SSSR, 269 (1983), pp. 543–547.

[24] M. Pereyra, *Proximal Markov chain Monte Carlo algorithms*, Statist. Comput., 26 (2016), pp. 745–760.
[25] J.-C. Pesquet and A. Repetti, *A class of randomized primal-dual algorithms for distributed optimization*, J. Nonlinear Convex Anal., 16 (2015), pp. 2353–2490.

[26] L. J. Rendell, A. M. Johansen, A. Lee, and N. Whiteley, *Global consensus Monte Carlo*, J. Comput. Graph. Statist., 30 (2021), pp. 249–259.

[27] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*, Springer Texts in Statistics, Springer, New York, NY, 2. ed., softcover reprint of the hardcover 2. ed. 2004 ed., 2010.

[28] L. I. Rudin, S. Osher, and E. Fatemi, *Nonlinear total variation based noise removal algorithms*, Phys. D, 60 (1992), pp. 259–268.

[29] A. Salim and P. Richtárik, *Primal dual interpretation of the proximal stochastic gradient langevin algorithm*, in Proc. Ann. Conf. Neur. Inform. Proc. Syst., vol. 33, 2020, pp. 3786–3796.

[30] P.-A. Thouvenin, A. Abdulaziz, A. Dabbech, A. Repetti, and Y. Wiaux, *Parallel faceted imaging in radio interferometry via proximal splitting (faceted hypersara): I. algorithm and simulations*, to appear in Monthly Notices of the Royal Astronomical Society, (2022).

[31] P.-A. Thouvenin, A. Repetti, and P. Chainais, *A versatile distributed MCMC algorithm for large scale inverse problems*, in European Signal Processing Conference (EUSIPCO), Belgrade, Serbia, 2022. to appear.

[32] P. Tseng, *Convergence of a block coordinate descent method for nondifferentiable minimization*, J. Optim. Theory Appl., 109 (2001), pp. 475–494.

[33] M. Vono, N. Dobigeon, and P. Chainais, *Bayesian image restoration under poisson noise and log-concave prior*, in ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, United Kingdom, 2019, pp. 1712–1716.

[34] M. Vono, N. Dobigeon, and P. Chainais, *Split-and-augmented Gibbs sampler - application to large-scale inference problems*, IEEE Transactions on Signal Processing, 67 (2019), pp. 1648–1661.

[35] M. Vono, N. Dobigeon, and P. Chainais, *Asymptotically exact data augmentation: Models, properties, and algorithms*, J. Comput. Graph. Statist., 30 (2021), pp. 335–348.

[36] B. C. Vů, *A splitting algorithm for dual monotone inclusions involving cocoercive operators*, Advances in Computational Mathematics, 38 (2013), pp. 667–681.

[37] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, *Image quality assessment: from error visibility to structural similarity*, IEEE Trans. Image Process., 13 (2004), pp. 600–612.