A Distributed Block-Split Gibbs Sampler with Hypergraph Structure for High-Dimensional Inverse Problems

Pierre-Antoine Thouvenin, Audrey Repetti, Pierre Chainais

A Distributed Block-Split 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 Split Gibbs sampler (SGS) to efficiently solve such problems involving 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 SGS. 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. Supplementary material to reproduce the experiments is available online.

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

[*]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

This work focuses on sampling from a generic distribution of the form

$$\pi(\boldsymbol{x}) \propto \exp(-h(\boldsymbol{x}) - f(\boldsymbol{x}) - g(\boldsymbol{D}\boldsymbol{x})), \tag{1.1}$$

where $h\colon \mathbf{H} \to]-\infty, +\infty]$ is a Lipschitz-differentiable function, $f\colon \mathbf{H} \to]-\infty, +\infty]$ and $g\colon \mathbf{G} \to]-\infty, +\infty]$ are possibly non-smooth functions, and $\boldsymbol{D}\colon \mathbf{H} \to \mathbf{G}$ is a linear operator. Such a distribution typically arises as the posterior distribution involved in imaging inverse problems, from which Bayesian estimators need to be formed [26]. In the remainder, we will consider that $\mathbf{H} = \mathbb{R}^{\overline{N}}$ and $\mathbf{G} = \mathbb{R}^{\overline{M}}$, where $\overline{N}$ and $\overline{M}$ are very large. Sampling from (1.1) is challenging due to (i) the presence of the composite function $g \circ \boldsymbol{D}$ and (ii) the large dimension of $\mathbf{H}$ and $\mathbf{G}$. To address these issues, this paper proposes a distributed MCMC algorithm which (i) leverages the approximate augmentation AXDA [37] to decouple (*split*) functions involved in (1.1), and (ii) exploits the structure of (1.1) to design a distributed sampler reminiscent of block-coordinate approaches in optimization. We briefly discuss recent splitting-based distributed samplers in Section 1.1, and highlight some of their limitations. Scalable splitting optimization approaches are reviewed in Section 1.2 to motivate this paper. The proposed approach, which takes further inspiration from the optimization literature, is outlined in Section 1.3.

1.1 Sampling methods: splitting and distributed techniques

Markov chain Monte Carlo (MCMC) algorithms are generic approaches providing estimates with associated credibility intervals [29]. They aim to generate a Markov chain that yields samples from the target distribution (1.1) in the stationary regime. Nevertheless, they are often considered computationally too expensive to handle high dimensional problems, especially when composite functions are involved. This is often the case with inverse problems in image processing. Over the last decade, many authors have proposed more versatile and scalable optimization-inspired MCMC algorithms [16, 25, 31]. These approaches exploit quantities repeatedly used in optimization to efficiently explore high dimensional parameter spaces, most often gradients and proximal operators[1].

A splitting approach based on an *asymptotically exact data augmentation* (AXDA) has also recently been proposed by [36, 37]. Inspired by splitting optimization approaches [20], AXDA introduces auxiliary variables to split composite distributions. The density (1.1) is then approximated by

$$\pi_{(\alpha,\beta)}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}) \propto \exp\big(-h(\boldsymbol{x}) - f(\boldsymbol{x}) - g(\boldsymbol{z}) - \phi_\alpha(\boldsymbol{D}\boldsymbol{x}, \boldsymbol{z} - \boldsymbol{u}) - \psi_\beta(\boldsymbol{u})\big), \tag{1.2}$$

[1]The proximal operator of a proper, lower semi-continuous function $f\colon \mathbb{R}^N \to]-\infty, +\infty]$ is defined for any $\boldsymbol{y} \in \mathbb{R}^N$ by [18]: $\operatorname{prox}_f(\boldsymbol{y}) = \underset{\boldsymbol{x}\in\mathbb{R}^N}{\operatorname{argmin}} \ \{f(\boldsymbol{x}) + \|\boldsymbol{x} - \boldsymbol{y}\|_2^2/2\}$.

where $\phi_\alpha \colon \mathbf{G} \times \mathbf{G} \to]-\infty, +\infty]$, $\psi_\beta \colon \mathbf{G} \to]-\infty, +\infty]$, α controls the discrepancy between $\boldsymbol{D}\boldsymbol{x}$ and $\boldsymbol{z} - \boldsymbol{u}$, and β is an augmentation parameter. The variable $\boldsymbol{z}$ is to be interpreted as a splitting variable, and $\boldsymbol{u}$ is an additional augmentation parameter. The role of ϕ_α is to strongly couple $\boldsymbol{D}\boldsymbol{x}$ and $\boldsymbol{z} - \boldsymbol{u}$, while ψ_β keeps $\boldsymbol{u}$ small enough. This latter parameter is not mandatory, but improves the mixing properties of the sampler by a further decoupling between $\boldsymbol{D}\boldsymbol{x}$ and $\boldsymbol{z}$ [36]. For appropriate choices of ϕ_α and ψ_β, the marginal distribution of $\boldsymbol{x}$ with respect to (1.2) converges to the target distribution (1.1) as $(\alpha, \beta) \to (0, 0)$. A Gibbs sampler is proposed in [36, 37] to draw samples from (1.2), referred to as the split Gibbs sampler (SGS). The additional cost of adding new variables is compensated by the benefit of this divide-to-conquer strategy. Further details on this splitting technique are provided in Section 2.1. Designing efficient algorithms to handle distributions of the form (1.1) becomes even more challenging when potentially many composite functions are considered. An extension of SGS has been considered [36, 28] for distributions involving $C \in \mathbb{N}^*$ composite terms, with a density of the form

$$\pi(\boldsymbol{x}) \propto \exp\big(- h(\boldsymbol{x}) - f(\boldsymbol{x}) - \sum_{c=1}^{C} g_c(\boldsymbol{D}_c\boldsymbol{x})\big), \tag{1.3}$$

where for every $c \in \{1, \dots, C\}$, $\mathbf{G}_c = \mathbb{R}^{\overline{M}_c}$, $\overline{M} = \sum_{c=1}^{C} \overline{M}_c$, $\boldsymbol{D}_c \colon \mathbf{H} \to \mathbf{G}_c$ and $g_c \colon \mathbf{G}_c \to]-\infty, +\infty]$. Applying AXDA to (1.3) leads to an approximation with density

$$\begin{aligned}\pi_{(\boldsymbol{\alpha},\boldsymbol{\beta})}\big(\boldsymbol{x}, (\boldsymbol{z}_c, \boldsymbol{u}_c)_{1\leqslant c\leqslant C}\big) \\ \propto \exp\Big(- h(\boldsymbol{x}) - f(\boldsymbol{x}) - \sum_{c=1}^{C} \Big(g_c(\boldsymbol{z}_c) + \phi_{c,\alpha_c}(\boldsymbol{D}_c\boldsymbol{x}, \boldsymbol{z}_c - \boldsymbol{u}_c) + \psi_{c,\beta_c}(\boldsymbol{u}_c)\Big)\Big),\end{aligned} \tag{1.4}$$

where $(\boldsymbol{z}_c, \boldsymbol{u}_c)_{1\leqslant c\leqslant C}$ are auxiliary variables and, for $c \in \{1, \dots, C\}$, $\phi_{c,\alpha_c} \colon \mathbf{G}_c \times \mathbf{G}_c \to]-\infty, +\infty]$ and $\psi_{c,\beta_c} \colon \mathbf{G}_c \to]-\infty, +\infty]$. The variables $(\boldsymbol{z}_c, \boldsymbol{u}_c)_{1\leqslant c\leqslant C}$ are conditionally independent, paving the way to a distributed implementation on a client-server architecture.

Only a few distributed samplers have been proposed in the literature [28, 36, 37]. However, these samplers only focus on distributing the splitting variables associated with the composite functions $g_c \circ \boldsymbol{D}_c$ from (1.3), without decomposing the global variable $\boldsymbol{x}$ into blocks. In particular, [28] propose a consensus-based approximate posterior distribution, addressed with a distributed Metropolis-within-Gibbs sampler relying on a *client-server* architecture. The sampler exploits the conditional independence between the splitting 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, e.g., for distributed logistic regression. However, this setting has several drawbacks when turning to high dimensional problems. First, the consensus constraint necessitates to duplicate the variables of interest on the different workers. 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 separates composite functions only, without splitting the high dimensional global variable of interest $\boldsymbol{x}$ into blocks. In high dimensions, it may be of interest to split the variable $\boldsymbol{x}$ itself, not only the data or the composite functions.

To summarize, splitting techniques have been introduced in sampling methods. They have permitted to handle multiple composite functions in parallel. Distributed versions have also been proposed in the literature, but restricted to client-server approaches. The client-server architecture can be critical in a high dimensional setting. In addition, none of these methods considered splitting the high-dimensional variable of interest $\boldsymbol{x}$ into blocks. Block-coordinate approaches can be necessary in practice to handle high-dimensional problems, e.g., when $\boldsymbol{x}$ is an image with more than 10^6 unknown parameters. Since the distributed split Gibbs sampler proposed in this work are essentially inspired by optimization approaches, the next paragraph reviews methods from the optimization literature that combine all at once composite-function splitting, variable splitting (i.e., block-coordinate approaches) and distributed techniques.

1.2 Splitting, distributed and block-coordinate methods in optimization

Optimization-based inference consists in estimating a mode of the distribution (1.1) (e.g., the maximum *a posteriori* (MAP) estimator), defined as a solution to

$$\underset{\boldsymbol{x}\in\mathbf{H}}{\text{minimize}} \;\; h(\boldsymbol{x}) + f(\boldsymbol{x}) + g(\boldsymbol{D}\boldsymbol{x}). \tag{1.5}$$

Optimization algorithms aim to build sequences that asymptotically converges to a solution to problem (1.5). Problem (1.5) can be efficiently solved with proximal primal-dual methods [20, 13, 38], that can be seen as the optimization counterpart of the splitting approaches discussed in Section 1.1. Similarly to AXDA, primal-dual methods rely on an auxiliary variable $\boldsymbol{z} \in \mathbf{G}$ associated with the composite function $g \circ \boldsymbol{D}$. The variable $\boldsymbol{x} \in \mathbf{H}$ is referred to as the *primal variable* while $\boldsymbol{z}$ is called the *dual variable*, since it is associated with the dual space $\mathbf{G}$ induced by the operator $\boldsymbol{D}$. It is worth noticing that the SGS approach developed by [36] was directly inspired by such an optimization splitting technique. Splitting proximal algorithms benefit from many acceleration techniques (e.g., inertia [2, 24], 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, 20].

Designing efficient algorithms to handle distributions of the form (1.3) becomes more challenging when multiple composite functions are involved, especially when the dimensions of $\boldsymbol{x}$ and $\boldsymbol{y}$ increase. In this context, the counterpart of problem (1.3) is given by

$$\underset{\boldsymbol{x}\in\mathbf{H}}{\text{minimize}} \;\; h(\boldsymbol{x}) + f(\boldsymbol{x}) + \sum_{c=1}^{C} g_c(\boldsymbol{D}_c\boldsymbol{x}). \tag{1.6}$$

Primal-dual proximal algorithms [13, 38, 20, 27, 9] permit to handle composite terms in (1.6) in parallel in the dual domains induced by operators $(\boldsymbol{D}_c)_{1\leqslant c\leqslant C}$. This is possible since dual variables can be distributed on multiple workers.

Further, to address problems with high dimensional variable $\boldsymbol{x}$, a usual strategy in optimization consists in splitting $\boldsymbol{x}$ into blocks $(\boldsymbol{x}_k)_{1\leqslant k\leqslant K}$, and either alternate between the blocks [4, 8, 22, 34],

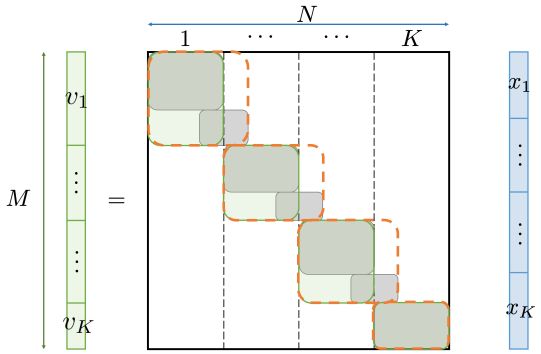


Figure 1: Example of the block-sparse structure of the matrix $\boldsymbol{D}$ involved in (1.1). The columns of $\boldsymbol{D}$ are split into K contiguous blocks (black dashed lines), with a small overlap compared to the size of the blocks. The orange dashed rectangles highlight the subparts of $\boldsymbol{D}$ implemented on each worker k. These act on a parameter block stored on worker k, and a few parameters stored on worker $k+1$.

or distribute the blocks over multiple workers [11, 27]. In particular, [27] combine such block-coordinate approaches with primal-dual splitting techniques to parallelize and distribute both the primal and dual variables. Then, the associated minimization problem is of the form

$$\underset{\boldsymbol{x}=(\boldsymbol{x}_k)_{1\leqslant k\leqslant K}\in\mathbf{H}}{\text{minimize}} \quad \sum_{k=1}^{K} h_k(\boldsymbol{x}_k) + f_k(\boldsymbol{x}_k) + \sum_{c=1}^{C} g_c(\sum_{k=1}^{K} \boldsymbol{D}_{c,k}\boldsymbol{x}_k), \tag{1.7}$$

where, for every $k \in \{1,\dots,K\}$, h_k and f_k only act on the k-th block of the variable $\boldsymbol{x} \in \mathbf{H}$. Such a formulation is of particular interest when considering block-sparse matrices $\boldsymbol{D}_c$, as illustrated in Figure 1. Finally, [27] also developed asynchronous distributed algorithms over hypergraphs[2] structures by combining the resulting block-coordinate primal-dual algorithms with consensus constraints that impose all $(\boldsymbol{x}_k)_{1\leqslant k\leqslant K}$ to be equal. This strategy provides a high flexibility in the choice of the distribution architecture. Section 3.2 will detail the interest of hypergraphs. Unfortunately, these algorithms only provide a point estimate, without additional information. In absence of ground truth, these approaches do not directly quantify the uncertainty over the estimate. The proposed approach will focus on drawing samples from the distribution corresponding to (1.7) using a distributed architecture.

1.3 Proposed distributed block-coordinate SGS

This work focuses on cases where $\boldsymbol{x}$ belongs to a high dimensional space, for instance for $\overline{N}$ larger than 10^6. We introduce a distributed block-coordinate SGS to sample from (1.3) by splitting the global variable of interest $\boldsymbol{x}$ into blocks $(\boldsymbol{x}_k)_{1\leqslant k\leqslant K}$. This approach will also be able to handle multiple composite functions as in (1.7). To this aim, we pair SGS with optimization-inspired MCMC transition kernels [16, 25, 31] to sample from conditional distributions that would otherwise be either intractable or challenging to handle in a distributed setting.

[2]Hypergraphs generalize structures of graphs, where edges can connect multiple nodes (i.e., variables) together, hence generalizing communications between variables.

Some steps of the MCMC sampler will correspond to term-wise operations, which are easy to distribute. Some others will involve the linear operator $\boldsymbol{D}$ and its adjoint $\boldsymbol{D}^*$, calling for communications. This work is aimed at designing an efficient distributed version of the corresponding serial sampler, in particular by dealing with terms that involve $\boldsymbol{D}$ or $\boldsymbol{D}^*$.

The distributed structure associates each block $\boldsymbol{x}_k$ with a worker k. Communications between the workers are governed by the structure of the linear operators $(\boldsymbol{D}_c)_{1\leqslant c\leqslant C}$. Precisely, the notion of hypergraph will be used to deal with this aspect. Section 4 is dedicated to technical details. For the sake of clarity and simplicity of the presentation, the main part of the paper presents the proposed distributed algorithm with $C = 1$ in (1.3), corresponding to the target distribution (1.1). The appendix extends the proposed approach to distributions of the form (1.3) with $C > 1$. We also focus on the case with augmentation variables as in (1.2) to ensure better mixing properties [36]. Note that our approach can be used without this optional variable.

The 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, with no central server. An SPMD architecture can drastically reduce the communication costs compared to a client-server architecture when a small number of workers is involved in each communication channel. This is especially the case when the structure of $\boldsymbol{D}$ induces *localized* couplings between parameters. Figure 1 illustrates the particular case of a block-sparse matrix $\boldsymbol{D}$: localized interactions between parameters imply a block-sparse structure which induces a hypergraph structure, see Section 4 for details. This case is often encountered in practice, e.g., for inverse problems in imaging, with applications such as image deconvolution or inpainting, or when considering models based on a TV norm regularization. When multiple composite terms are considered as in (1.7), a parallel implementation based on a client-server architecture would be possible by exploiting conditional independence between blocks of variables. However this configuration may suffer from communication bottlenecks, with a number of workers limited by the number of conditionally independent blocks of variables in the model. In contrast, the SPMD approach permits to use a larger number of workers by exploiting the structure of the hypergraphs induced by the structure of operators $\boldsymbol{D}_c$ involved.

The remainder of the paper is organized as follows. The AXDA approach and the SGS algorithm [36, 37] are recalled in Section 2. An overview of the proposed distributed SGS is given in Section 3. This section further outlines how limitations of the client-server architecture can be addressed by a fully decentralized SPMD architecture. The advantages offered by the latter are specifically emphasized for distributions defined on a high-dimensional space, e.g., in imaging inverse problems. The proposed hypergraph model is introduced in Section 4, and the associated distributed block-coordinate SGS is given in Section 5. Section 6 describes an SPMD implementation of the proposed method for a large scale image deconvolution problem. Conclusion and perspectives are given in Section 7. Eventually, the proposed method is extended to the general case of distributions involving multiple composite terms in Appendix A.

2 AXDA approach and the split Gibbs samplers

In this section, we summarize the approximation results at the basis of the AXDA approach [36]. We also describe a Gibbs sampler to approximately draw samples from (1.1), referred to as the Split Gibbs sampler (SGS) [37].

2.1 AXDA splitting approach

A Gibbs sampler can be used to draw samples from (1.2), approximating the target distribution (1.1). The sampler successively draws one sample from each conditional distribution

$$\pi_\alpha\left(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{u}\right) \propto \exp\big(-h(\boldsymbol{x}) - f(\boldsymbol{x}) - \phi_\alpha(\boldsymbol{D}\boldsymbol{x}, \boldsymbol{z}-\boldsymbol{u})\big), \tag{2.1}$$

$$\pi_\alpha\left(\boldsymbol{z} \mid \boldsymbol{v}, \boldsymbol{u}\right) \propto \exp\big(-g(\boldsymbol{z}) - \phi_\alpha(\boldsymbol{v}, \boldsymbol{z}-\boldsymbol{u})\big), \tag{2.2}$$

$$\pi_{(\alpha,\beta)}\left(\boldsymbol{u} \mid \boldsymbol{v}, \boldsymbol{z}\right) \propto \exp\big(-\phi_\alpha(\boldsymbol{v}, \boldsymbol{z}-\boldsymbol{u}) - \psi_\beta(\boldsymbol{u})\big), \tag{2.3}$$

where $\boldsymbol{v} = \boldsymbol{D}\boldsymbol{x}$. The SGS associated with (2.1)–(2.3) is reported in Algorithm 1.

Algorithm 1: Generic Split Gibbs Sampler (SGS) [36].

Input: $\boldsymbol{z}^{(0)} \in \mathbf{G}$, $\boldsymbol{u}^{(0)} \in \mathbf{G}$, $(\alpha, \beta) \in]0, +\infty[^2$

for $t = 0$ **to** T **do**

- $\boldsymbol{x}^{(t)} \sim \pi_\alpha\left(\boldsymbol{x} \mid \boldsymbol{z}^{(t)}, \boldsymbol{u}^{(t)}\right)$,
- $\boldsymbol{v}^{(t)} = \boldsymbol{D}\boldsymbol{x}^{(t)}$,
- $\boldsymbol{z}^{(t+1)} \sim \pi_\alpha\left(\boldsymbol{z} \mid \boldsymbol{v}^{(t)}, \boldsymbol{u}^{(t)}\right)$,
- $\boldsymbol{u}^{(t+1)} \sim \pi_{(\alpha,\beta)}\left(\boldsymbol{u} \mid \boldsymbol{v}^{(t)}, \boldsymbol{z}^{(t+1)}\right)$

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

Under technical conditions on the functions ϕ_α and ψ_β in (1.2), [36] showed that the marginal distribution of $\boldsymbol{x}$ can be made arbitrarily close to the original distribution $\pi\left(\boldsymbol{x}\right)$ given in (1.1), typically when $(\alpha, \beta) \to (0, 0)$. This result is summarized in the following proposition.

Proposition 2.1 *[36, Thm. 1 and 2] Let $(\alpha, \beta) \in]0, +\infty[^2$, $\phi_\alpha : \mathbf{G} \times \mathbf{G} \to]-\infty, +\infty]$ and $\psi_\beta : \mathbf{G} \to]-\infty, +\infty]$. Assume that*

$$(\forall \boldsymbol{x} \in \mathbf{H}) \quad \lim_{\alpha \to 0} \frac{\exp\big(-\phi_\alpha(\boldsymbol{D}\boldsymbol{x}, \boldsymbol{z})\big)}{\int_{\mathbf{G}} \exp\big(-\phi_\alpha(\boldsymbol{D}\boldsymbol{x}, \boldsymbol{z})\big) d\boldsymbol{z}} = \delta_{\boldsymbol{x}}(\boldsymbol{z}), \tag{2.4}$$

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

$$(\forall (\boldsymbol{x}, \boldsymbol{z}) \in \mathbf{H} \times \mathbf{G}) \quad \int_{\mathbf{G}} \exp\big(-\phi_\alpha(\boldsymbol{D}\boldsymbol{x}, \boldsymbol{z}-\boldsymbol{u}) - \psi_\beta(\boldsymbol{u})\big) d\boldsymbol{u} \propto \exp\big(-\phi_{\eta_{\alpha,\beta}}(\boldsymbol{D}\boldsymbol{x}, \boldsymbol{z})\big). \tag{2.5}$$

Let, for $\boldsymbol{x} \in \mathbf{H}$, $\pi_{\eta_{\alpha,\beta}}(\boldsymbol{x}) = \int_{\mathbf{G}\times\mathbf{G}} \pi_{\alpha,\beta}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}) d\boldsymbol{u} d\boldsymbol{z}$. Then, $\|\pi - \pi_{\eta_{\alpha,\beta}}\|_{TV} \to 0$ when $\eta_{\alpha,\beta} \to 0$.

Theorem 2.1 ensures that the marginal distribution of $\boldsymbol{x}$ with respect to $\pi_{(\alpha,\beta)}$ converges to the target distribution π given in (1.1), provided conditions (2.4) and (2.5) hold. In particular, in [36], these conditions are shown to be satisfied for $\eta^2_{\alpha,\beta} = \alpha^2 + \beta^2$, with

$$\begin{cases} \phi_\alpha(\boldsymbol{D}\boldsymbol{x}, \boldsymbol{z} - \boldsymbol{u}) = \frac{1}{2\alpha^2}\|\boldsymbol{D}\boldsymbol{x} - (\boldsymbol{z} - \boldsymbol{u})\|^2, \\ \psi_\beta(\boldsymbol{u}) = \frac{1}{2\beta^2}\|\boldsymbol{u}\|^2. \end{cases} \tag{2.6}$$

In the particular case when $f \equiv 0$, stronger results can be found in [37, Theorem 2], including theoretical guarantees on the convergence rate.

2.2 PSGLA within SGS

Drawing samples directly from the conditional distributions (2.1)–(2.3) 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.

To sample from (1.2), Metropolis-Hastings transition kernels are classical choices to draw samples from the conditional distributions $\pi_\alpha(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{u})$, $\pi_\alpha(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{u})$ and $\pi_{(\alpha,\beta)}(\boldsymbol{u} \mid \boldsymbol{x}, \boldsymbol{z})$, leading to a conditional Metropolis-Hastings sampler [19]. Appropriate proposals include Langevin-based kernels, such as the Moreau-Yosida unadjusted Langevin algorithm (MYULA) [16] and the proximal stochastic gradient Langevin algorithm (PSGLA) [31]. 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 while maintaining a good approximation of the target distribution by using unadjusted kernels, as in [16]. This approach is adopted in the following. Deriving non-asymptotic convergence bounds and analyzing the bias between the target distribution and (1.2) 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) will ensure that the transition kernels are amenable to a distributed implementation, see Section 4. This implementation will be adapted to the specific structure of the distributions $\pi_\alpha(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{u})$, $\pi_\alpha(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{u})$ and $\pi_{\alpha,\beta}(\boldsymbol{u} \mid \boldsymbol{x}, \boldsymbol{z})$. In particular, for the proposed distributed SGS, we consider the case where any appropriate transition kernel can be used to sample $(\boldsymbol{z}, \boldsymbol{u})$. The parameter $\boldsymbol{x}$ is approximately sampled using a PSGLA transition to avoid the extra cost induced by a Metropolis correction step. To this aim, we assume that h is λ_h-Lipschitz-differentiable. For simplicity, ϕ_α and ψ_β are also taken as ℓ_2 norms as in (2.6), so that (2.4) and (2.5) are satisfied. Then $\boldsymbol{u}$ can be directly sampled from its Gaussian conditional distribution. These blanket assumptions, summarized in Theorem 2.2, will be adopted in the following.

Assumption 2.2

(i) ϕ_α and ψ_β are given by

$$(\forall (\boldsymbol{v}, \boldsymbol{u}) \in \mathbf{G}^2) \quad \phi_\alpha(\boldsymbol{v}, \boldsymbol{u}) = \frac{1}{2\alpha^2}\|\boldsymbol{v} - \boldsymbol{u}\|^2, \tag{2.7}$$

$$\psi_\beta(\boldsymbol{u}) = \frac{1}{2\beta^2}\|\boldsymbol{u}\|^2. \tag{2.8}$$

(ii) $h\colon \mathbf{H} \to \mathbb{R}$ is λ_h-Lipschitz differentiable, with $\lambda_h > 0$.

Algorithm 2: Proposed sampler (PSGLA within SGS).

Input: $\boldsymbol{x}^{(0)} \in \mathbf{H}$, $(\boldsymbol{z}^{(0)}, \boldsymbol{u}^{(0)}) \in \mathbf{G}^2$, $(\alpha, \beta) \in]0, +\infty[^2$, $\gamma \in \left]0, \left(\lambda_h + \|\boldsymbol{D}\|^2/\alpha^2\right)^{-1}\right[$

1 $\boldsymbol{v}^{(0)} = \boldsymbol{D}\boldsymbol{x}^{(0)}$;

2 **for** $t = 0$ **to** T **do**

// Draw $\boldsymbol{x}^{(t+1)}$ with a PSGLA [31] kernel

3 $\boldsymbol{x}^{(t+1)} = \mathrm{prox}_{\gamma f}\Big(\boldsymbol{x}^{(t)} - \gamma\nabla h(\boldsymbol{x}^{(t)}) - \gamma\boldsymbol{D}^*\Big(\nabla\phi_\alpha(\cdot, \boldsymbol{z}^{(t)} - \boldsymbol{u}^{(t)})(\boldsymbol{v}^{(t)})\Big) + \sqrt{2\gamma}\,\boldsymbol{w}^{(t)}\Big)$;

4 $\boldsymbol{v}^{(t+1)} = \boldsymbol{D}\boldsymbol{x}^{(t+1)}$;

// Draw $\boldsymbol{z}^{(t+1)}$ and $\boldsymbol{u}^{(t+1)}$ from their conditional distribution

5 $\boldsymbol{z}^{(t+1)} \sim \pi_\alpha\left(\boldsymbol{z} \mid \boldsymbol{v}^{(t+1)}, \boldsymbol{u}^{(t)}\right)$;

6 $\boldsymbol{u}^{(t+1)} \sim \mathcal{N}\Big(\frac{\beta^2}{\alpha^2+\beta^2}(\boldsymbol{z}^{(t+1)} - \boldsymbol{v}^{(t+1)}), \frac{\alpha^2+\beta^2}{\alpha^2\beta^2}\mathbf{I}\Big)$;

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

Theorem 2.2(i) ensures that Theorem 2.1 is valid, see (2.6). Theorem 2.2(ii) is necessary to use the PSGLA transition kernel in the proposed SGS detailed in Algorithm 2, where $\boldsymbol{D}^*$ denotes the adjoint of $\boldsymbol{D}$, and $(\boldsymbol{w}^{(t)})_{1\leqslant t\leqslant T}$ is a sequence of independent and identically distributed (i.i.d) standard Gaussian random variables in $\mathbf{H}$.

3 Proposed distributed SGS in a nutshell

For high-dimensional problems, every step of Algorithm 1 is computationally expensive, calling for a distributed algorithm. Steps 5 and 6 are easy to distribute since they correspond to term-wise operations. In contrast, steps 1, 3 and 4 require communications due to the presence of the linear operator $\boldsymbol{D}$ and its adjoint $\boldsymbol{D}^*$. This work is aimed at designing an efficient distributed version of Algorithm 2 by exploiting the hypergraph structure that can emerge from the topology of $\boldsymbol{D}$. This section describes the proposed approach *in a nutshell*, reducing technicality. Section 4 introduces the model to describe the hypergraph structure and the resulting distribution strategy. Section 5 details the distributed version of the proposed Algorithm 3.

In Section 3.1, we first explain the advantages offered by an SPMD distributed implementation over a client-server approach to address high dimensional problems. To enable this SPMD implementation, we rely on hypergraph structures that facilitate flexible communications. These

hypergraph structures are reminiscent of the topology of $\boldsymbol{D}$, as described in Section 3.2. Practical considerations for an efficient distributed implementation are given in Section 3.3. We refer the reader to Section 4 and Section 5 for a rigorous description of the proposed distributed algorithm. For an example application of the proposed approach to an imaging inverse problem, the reader can jump to Section 6.

3.1 Splitting-based samplers

Splitting-based samplers from the literature [37, 28] can benefit from a distributed implementation on a client-server architecture. These algorithms can accommodate densities of the form (1.4) with $C > 1$ linear operators. After splitting, a collection of $K - 1 \leqslant C$ workers (i.e., *clients*) handle computations associated with groups of variables conditionally independent from one worker to another. The server handles operations on the full variable $\boldsymbol{x}$, possibly duplicated across all the workers. This configuration can suffer from several drawbacks listed below.

(i) **Shared variable and communication costs** : The computation of $(\boldsymbol{D}_c\boldsymbol{x})_{1\leqslant c\leqslant C}$ on the server can be expensive. The shared parameter $\boldsymbol{x}$ needs to be broadcast to all the clients at each iteration. Communications can significantly increase for applications defined on a high dimensional parameter space, such as inverse problems in imaging applications.

(ii) **Limited number of workers** : The number of workers K is restricted by the number of composite functions C since $K - 1 \leqslant C$, typically with $1 \leqslant C \leqslant 5$ for most imaging inverse problems. A client-server approach cannot be applied to distributions with $C = 1$ such as (1.1).

(iii) **Load balancing**: Computing costs induced by the operators $(\boldsymbol{D}_c)_{1\leqslant c\leqslant C}$ can be hard to balance over the workers. Prohibitive idle time due to synchronization with *stragglers* may thus drastically limit the parallel efficiency of the algorithm.

To address these issues, a possible approach consists in exploiting the hypergraph structure of the operator $\boldsymbol{D}$, as outlined in Section 3.2. Such an approach enables the use of an SPMD architecture [15], so that all the workers can conduct locally the same tasks over a subpart of the variables only. In this configuration, load balancing is in general easier to handle, with no restriction on the number of workers K imposed by the model and lighter communications as explained in Sections 3.2 and 3.3. Section 5.3 explains in more details how the SPMD architecture takes advantage of the hypergraph structure of the algorithm detailed in Sections 4 and 5.

3.2 A hypergraph structure to better communicate

We consider a localized linear operator $\boldsymbol{D}$ in (1.1), such that couplings between latent parameters are localized. This is for instance the case when $\boldsymbol{D}$ is block-sparse, see Figure 1. Such a structure

is not strictly 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. Practical uses of interest for block-sparse structures include imaging inverse problems, e.g., where $\boldsymbol{D}$ corresponds to a deconvolution or inpainting measurement operator. Block-sparse operators also appear when considering models such as the TV norm.

The structure of the matrix $\boldsymbol{D}$ 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 $\mathcal{H}$ is a pair $(\boldsymbol{x}, \boldsymbol{e})$ made of vertices $\boldsymbol{x} = (x_n)_{1 \leqslant n \leqslant N}$ and hyperedges $\boldsymbol{e} = (e_m)_{1 \leqslant m \leqslant M}$, where each hyperedge is a set of connected vertices, that is a subset of the $(x_n)_{1 \leqslant n \leqslant N}$. The main idea is to take benefit from this structure to distribute the computation of various quantities in Algorithm 2 over K workers. It appears that the product $\boldsymbol{D}\boldsymbol{x}$ is of special interest.

Figure 1 illustrates the proposed approach. The variable $\boldsymbol{x}$ is divided into K blocks $(\boldsymbol{x}_k)_{1 \leqslant k \leqslant K}$, with each block $\boldsymbol{x}_k$ assigned to a single worker. The hyperedges will characterize the necessary communications between the workers. A worker $k \in \{1, \dots, K\}$ stores $\boldsymbol{x}_k$ but it may as well need access to some coefficients of $\boldsymbol{x}_{k'}$ stored on another worker $k' \neq k$ to carry out its computations. Note that the operator $\boldsymbol{D}$ will be split once and for all in $\boldsymbol{D} = (D_{m,n})_{1 \leqslant m \leqslant M,\, 1 \leqslant n \leqslant N}$ in an adequate manner over the K workers as well; this decomposition will not necessitate any additional communication between the workers. The output quantity $\boldsymbol{v} = (v_m)_{1 \leqslant m \leqslant M} = \boldsymbol{D}\boldsymbol{x}$ computed in Algorithm 2 will also be distributed among the K workers. To this aim, each hyperedge $m \in \{1, \dots, M\}$ is associated to a worker denoted by $k_m \in \{1, \dots, K\}$. 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 $\boldsymbol{D}$ will be necessary to compute v_m. Communications occur when the worker $k = k_m$ requires a subpart of $\boldsymbol{x}_{k'}$ stored on another worker $k' \neq k_m$ to carry out the computation of v_m. A subset of each $\boldsymbol{x}_k$ will never be involved in communications since it is used to compute v_m only, and will not be used to compute any other $v_{m'}$. Finally, the worker $k = k_m$, that is in charge of the computation of v_m, will also store $\boldsymbol{x}_k$ to reduce communications. In the particular case when $\boldsymbol{D}$ is block-sparse, the communication cost will remain small as long as the subpart of $\boldsymbol{x}_{k'}$ required by worker k_m remains small.

3.3 Conditions for an efficient distributed implementation

Even though the proposed approach detailed in Sections 4 and 5 is very general, some particular structures of matrix $\boldsymbol{D}$ can ensure limited communications, which are often the bottleneck of distributed methods.

Complexity costs of algorithms can roughly be divided into three terms. First, a computation term $\tau_{\texttt{flop}}$ reflects the time to perform a single floating point operation. Second, a communication term $\tau_{\texttt{bandwidth}}$, defined as the inverse of the communication bandwidth, quantifies the time necessary to send a single value. Third, a latency term $\tau_{\texttt{latency}}$ represents the cost incurred by

establishing a communication. In practice, $\tau_{\texttt{flop}} \ll \tau_{\texttt{bandwidth}} \ll \tau_{\texttt{latency}}$. 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 SGS, both in terms of computations and communications, we can identify two conditions on the structure of $\boldsymbol{D}$. These conditions are typically satisfied when $\boldsymbol{D}$ 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 number 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 $(k, k') \in \{1, \dots, K\}^2$, $k \neq k'$, the subpart of $\boldsymbol{x}_k$ that need to be sent from k to k' needs to be small compared to $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k'}$.

4 Model and hypergraph structure

This section provides the notation and a description of the hypergraph model used to define the proposed distributed SGS. It gives a formal description of the proposed distributed computing strategy, which exploits the structure of the linear operator $\boldsymbol{D}$ and a separability assumption on the functions involved in (1.2). Notation is summarized in Tables 1 and 2, and illustrated on a simple example in Figure 2.

4.1 Hypergraph structure of the model

Let $\mathbf{H} = \mathbb{R}^{\overline{N}}$ be such that $\mathbf{H} = \mathsf{H}_1 \times \ldots \times \mathsf{H}_N$, where for every $n \in \{1, \dots, N\}$, $\mathsf{H}_n = \mathbb{R}^{N_n}$ and $\overline{N} = \sum_{n=1}^{N} N_n$. An element of $\mathbf{H}$ is denoted by $\boldsymbol{x} = (x_n)_{1 \leqslant n \leqslant N}$, where, for every $n \in \{1, \dots, N\}$, $x_n \in \mathsf{H}_n$. Similarly, let $\mathbf{G} = \mathbb{R}^{\overline{M}}$ be such that $\mathbf{G} = \mathsf{G}_1 \times \ldots \times \mathsf{G}_M$, where for every $m \in \{1, \dots, M\}$, $\mathsf{G}_m = \mathbb{R}^{M_m}$ and $\overline{M} = \sum_{m=1}^{M} M_m$. Let $\boldsymbol{v} = (v_m)_{1 \leqslant m \leqslant M}$ be an element of $\mathbf{G}$ such that, for every $m \in \{1, \dots, M\}$, $v_m \in \mathsf{G}_m$.

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

Model 4.1 Let $\boldsymbol{D} \colon \mathbf{H} \to \mathbf{G}$ be such that $\boldsymbol{D} = (D_{m,n})_{1 \leqslant m \leqslant M, 1 \leqslant n \leqslant N}$, with, for every $(m, n) \in \{1, \dots, M\} \times \{1, \dots, N\}$, $D_{m,n} \colon \mathsf{H}_n \to \mathsf{G}_m$. Let $\mathcal{H} = (\boldsymbol{x}, \boldsymbol{e})$ be the hypergraph associated with $\boldsymbol{D}$, with N vertices $\boldsymbol{x}$, and M hyperedges denoted by $\boldsymbol{e} = (e_m)_{1 \leqslant m \leqslant M}$, such that

$$(\forall m \in \{1, \dots, M\}) \quad e_m = \big\{ n \in \{1, \dots, N\} \;\big|\; D_{m,n} \neq 0 \big\}, \tag{4.1}$$

Notation	Definition	Given by operator $\boldsymbol{D}$	Given by user
$n \in \{1, \dots, N\}$	Indices for vertices	✓	
$m \in \{1, \dots, M\}$	Indices for hyperedges	✓	
$\boldsymbol{e} = (e_m)_{1 \leqslant m \leqslant M}$	Hyperedges	✓	
$e_m \subset \{1, \dots, N\}$	Vertex indices in hyperedge m	✓	
$k \in \{1, \dots, K\}$	Indices for workers		✓
$\mathcal{R}_k \subset \{1, \dots, K\} \setminus \{k\}$	Indices of workers $k' \in \{1, \dots, K\} \setminus \{k\}$ from which worker k receives vertex values		✓
$\mathcal{S}_k \subset \{1, \dots, K\} \setminus \{k\}$	Indices of workers $k' \in \{1, \dots, K\} \setminus \{k\}$ to which worker k sends vertex values		✓
$\mathbb{V}_k \subset \{1, \dots, N\}$	Indices of vertices stored on worker k, such that $(\mathbb{V}_k)_{1 \leqslant k \leqslant K}$ is a partition of $\{1, \dots, N\}$	✓	✓
$k_m \in \{1, \dots, K\}$	Worker associated with m-th hyperedge e_m (chosen by the user). k_m must satisfy $e_m \cap \mathbb{V}_{k_m} \neq \varnothing$		✓
$\mathcal{W}_m \subset \{1, \dots, K\} \setminus \{k_m\}$	Set of all workers but k_m, containing vertices from e_m	✓	
$\overline{\mathcal{W}}_m \subset \{1, \dots, K\}$	$\overline{\mathcal{W}}_m = k_m \cup \mathcal{W}_m$	✓	
$\mathbb{V}_{(k,k')} \subset \mathbb{V}_{k'}$	Indices of vertices sent from worker k' to worker k	✓	✓
$\mathbb{V}_{\mathcal{W}_m} \subset \{1, \dots, N\} \setminus \mathbb{V}_{k_m}$	$\mathbb{V}_{\mathcal{W}_m} = \bigcup_{k' \in \mathcal{W}_m} \mathbb{V}_{(k_m,k')}$ the set of vertex indices that will be communicated to worker k_m from all workers $k' \in \mathcal{W}_m$	✓	✓
$\mathbb{V}_{\overline{\mathcal{W}}_m} \subset \{1, \dots, N\}$	$\mathbb{V}_{\overline{\mathcal{W}}_m} = \mathbb{V}_{k_m} \cup \mathbb{V}_{\mathcal{W}_m}$ the set of vertex indices necessary to perform computations associated with k_m	✓	✓
$\mathbb{E}_k \subset \{1, \dots, M\}$	Indices of hyperedges only containing vertices stored on worker k	✓	✓
$\mathbb{E}_{(k,k')} \subset \{1, \dots, M\}$	Indices of hyperedges containing vertices sent from worker k' to worker k	✓	✓
$\mathbb{E}_{\mathcal{R}_k} \subset \{1, \dots, M\}$	$\mathbb{E}_{\mathcal{R}_k} = \cup_{k' \in \mathcal{R}_k} \mathbb{E}_{(k,k')}$ set of all hyperedges containing vertices that will be communicated to worker k	✓	✓
$\overline{\mathbb{E}}_k \subset \{1, \dots, M\}$	$\overline{\mathbb{E}}_k = \mathbb{E}_k \cup \mathbb{E}_{\mathcal{R}_k}$, such that $(\overline{\mathbb{E}}_k)_{1 \leqslant k \leqslant K}$ is a partition of $\{1, \dots, M\}$	✓	✓

Table 1: Sets used to define the hypergraph structure of the problem and the distributed algorithm.

where $0_{\mathsf{G}_m \times \mathsf{H}_n}$ 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 $\boldsymbol{D}$. Precisely, the M rows of $\boldsymbol{D}$ 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 $m \in \{1, \dots, M\}$, the hyperedge e_m links nodes (x_n) if $D_{m,n} \neq 0_{\mathsf{G}_m \times \mathsf{H}_n}$. Hence, $\boldsymbol{D}$ can be seen as a weighted incidence matrix associated with $\mathcal{H}$. In this context, any variable in $\mathbf{G}$ can be seen as a hyperedge weight of the hypergraph $\mathcal{H}$.

According to Theorem 4.1, the computation of $\boldsymbol{v} = \boldsymbol{D}\boldsymbol{x}$ in Algorithm 2 can be decomposed and computed block-wise as

$$\boldsymbol{v} = (v_m)_{1 \leqslant m \leqslant M} \text{ with } (\forall m \in \{1, \dots, M\}) \quad v_m = \sum_{n \in e_m} D_{m,n} x_n. \tag{4.2}$$

In (4.2), for every $m \in \{1, \dots, M\}$, only the non-zero blocks of $\boldsymbol{D}$ are taken into account, i.e.,

Notation	Definition
$\boldsymbol{x} = (x_n)_{1 \leqslant n \leqslant N} \in \mathsf{H}$	Vertex values
$\boldsymbol{D} \colon \mathsf{H} \to \mathsf{G}$	$\boldsymbol{D} = (D_{m,n})_{1 \leqslant m \leqslant M, 1 \leqslant n \leqslant N}$ linear operator defining the hypergraph structure
$\boldsymbol{u}, \boldsymbol{v} \in \mathsf{G}$ $\boldsymbol{v} = (v_m)_{1 \leqslant m \leqslant M} \in \mathsf{G}$	Hyperedge weights
$\boldsymbol{v}_k \in \mathsf{G}_k$	$\boldsymbol{v}_k = (v_m)_{m \in \mathbb{E}_k}$ hyperedge weights stored on worker k
$\boldsymbol{x}_k \in \mathsf{H}_k$	$\boldsymbol{x}_k = (x_n)_{n \in \mathbb{V}_k}$ vertex values stored on worker k
$\boldsymbol{D}_{m,k} \colon \mathsf{H}_k \to \mathsf{G}_m$	$\boldsymbol{D}_{m,k} = (D_{m,n})_{n \in \mathbb{V}_k}$, for $m \in \mathbb{E}_k$, subpart of $\boldsymbol{D}$ stored on worker k, associated with hyperedges containing vertices only on worker k
$\boldsymbol{x}_{(k,k')} \in \bigtimes_{n \in \mathbb{V}_{(k,k')}} \mathsf{H}_n$	$\boldsymbol{x}_{(k,k')} = (x_n)_{n \in \mathbb{V}_{(k,k')}}$ vertex values sent from worker k' to worker k
$\boldsymbol{D}_{m,(k,k')} \colon \bigtimes_{n \in \mathbb{V}_{(k,k')}} \mathsf{H}_n \to \mathsf{G}_m$	$\boldsymbol{D}_{m,(k,k')} = (D_{m,n})_{n \in \mathbb{V}_{(k,k')}}$, for $m \in \mathbb{E}_{(k,k')}$, subpart of $\boldsymbol{D}$ stored on worker k, associated with hyperedges containing vertices overlapping workers k and k'
$\boldsymbol{x}_{\overline{\mathcal{W}}_m} \in \bigtimes_{n \in \mathbb{V}_{\overline{\mathcal{W}}_m}} \mathsf{H}_n$	$\boldsymbol{x}_{\overline{\mathcal{W}}_m} = (x_n)_{n \in \mathbb{V}_{\overline{\mathcal{W}}_m}}$, concatenation of vertex values stored on worker k_m, and those sent from all worker $k' \in \mathcal{W}_m$ to worker k_m
$\boldsymbol{D}_{m,\overline{\mathcal{W}}_m} \colon \bigtimes_{n \in \mathbb{V}_{\overline{\mathcal{W}}_m}} \mathsf{H}_n \to \mathsf{G}_m$	$\boldsymbol{D}_{m,\overline{\mathcal{W}}_m} = (D_{m,n})_{n \in \mathbb{V}_{\overline{\mathcal{W}}_m}}$, for $m \in \mathbb{E}_{\mathcal{R}_k}$, subpart of $(D_{m,n})_{1 \leqslant n \leqslant N}$ corresponding to vertices stored either on worker k_m or on a worker in $\mathcal{W}_m$

Table 2: Notation used for the variables involved in the proposed distributed algorithm.

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 $\boldsymbol{D}\boldsymbol{x}$, while limiting the communication cost.

4.2 Distribution of the hypergraph nodes over the workers

Let $K \in \mathbb{N}^*$, $K \leqslant N$, be the number of workers available to the user to parallelize the algorithm. On each worker $k \in \{1, \dots, K\}$, we store a subpart $\boldsymbol{x}_k$ of the nodes $\boldsymbol{x}$ of the hypergraph $\mathcal{H}$. The pattern for the distribution of the nodes $\boldsymbol{x}$ over the K workers will be driven by the topology of the matrix $\boldsymbol{D}$, as explained in Section 4.1. This pattern needs to be fixed before designing the algorithm. Note that the blocks $(D_{m,n})_{1 \leqslant m \leqslant M,\, n \in e_m}$ of $\boldsymbol{D}$ will be distributed over the workers as well, but will not require to be communicated. In particular, we provide below the notation to split the hypergraph (i.e., the operator $\boldsymbol{D}$) 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 $\boldsymbol{D}$.

4.2.1 Distributing the hypergraph over the workers

Let $(\mathbb{V}_k)_{1 \leqslant k \leqslant K}$ be the partition of the set of vertex indices $\{1, \dots, N\}$ such that, for every $k \in \{1, \dots, K\}$, $\mathbb{V}_k$ is the set of vertex indices handled on worker k, and for every $\boldsymbol{x} \in \mathsf{H}$, $\boldsymbol{x}_k =$

$(x_n)_{n\in\mathbb{V}_k}$. Recall that $k_m \in \{1,\dots,K\}$ denotes the worker associated with the m-th hyperedge e_m. The association of hyperedges to workers is a choice left to the user. Then, we must have $e_m \cap \mathbb{V}_{k_m} \neq \varnothing$, 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

$$\overline{\mathbb{E}}_k = \big\{m \in \{1,\dots,M\} \;\big|\; k_m = k\big\}. \tag{4.3}$$

Note that $\big(\overline{\mathbb{E}}_k\big)_{1\leqslant k\leqslant K}$ defines a partition of $\{1,\dots,M\}$ over the K workers. Optimizing the configuration for a specific operator $\boldsymbol{D}$ is, on its own, a resource allocation problem [21] that is out of the scope of this work. Using notation (4.3), for any $\boldsymbol{v} \in \mathbf{G}$, we can denote by $\boldsymbol{v}_k = (v_m)_{m\in\overline{\mathbb{E}}_k}$ the hyperegde weights stored on worker k.

We also define $\mathbb{E}_k$, the set of hyperedge indices only containing vertices stored on worker k, as

$$\mathbb{E}_k = \big\{m \in \{1,\dots,M\} \;\big|\; n \in e_m \;\Leftrightarrow\; n \in \mathbb{V}_k\big\}. \tag{4.4}$$

The sets $(\mathbb{E}_k)_{1\leqslant k\leqslant K}$ identify the rows of $\boldsymbol{D}$ 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 $m \in \overline{\mathbb{E}}_k \setminus \mathbb{E}_k$ 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 $m \in \{1,\dots,M\} \setminus (\bigcup_{k=1}^{K} \mathbb{E}_k)$.

For a fixed hyperedge e_m, let $\mathcal{W}_m \subset \{1,\dots,K\} \setminus \{k_m\}$ be the set of workers different from k_m containing vertices x_n belonging to the same hyperedge e_m, i.e.,

$$\mathcal{W}_m = \big\{k \in \{1,\dots,K\} \setminus \{k_m\} \;\big|\; \exists n \in e_m \text{ such that } n \in \mathbb{V}_k\big\}.$$

Using this notation, we can give an equivalent definition of (4.4) as, for every $k \in \{1,\dots,K\}$, $\mathbb{E}_k = \big\{m \in \{1,\dots,M\} \;\big|\; k_m = k \text{ and } \mathcal{W}_m = \varnothing\big\}$ (i.e., no overlap, no communication). For completeness, we also introduce the notation $\overline{\mathcal{W}}_m = k_m \cup \mathcal{W}_m$.

For every $m \in \{1,\dots,M\} \setminus (\bigcup_{k=1}^{K} \mathbb{E}_k)$, workers $k' \in \mathcal{W}_m$ will need to send some vertex values from $\boldsymbol{x}_{k'}$ to the worker k_m so that it can compute v_m, as described in (4.2). For every $k \in \{1,\dots,K\}$, we denote by $\mathcal{R}_k \subset \{1,\dots,K\} \setminus \{k\}$ the set of workers k' from which worker k receives vertices. These workers store vertices belonging to a hyperedge of index $m \in \{1,\dots,M\}$ such that $k_m = k$, but that are not stored on worker k. Similarly, we denote by $\mathcal{S}_k \subset \{1,\dots,K\}\setminus\{k\}$ the set of workers k' to which worker k is sending vertices, i.e., all the workers k' such that $k \in \mathcal{R}_{k'}$.

Communications will occur through hyperedges connecting different workers. For every $k' \in \mathcal{R}_k$, let

$$\mathbb{E}_{(k,k')} = \big\{m \in \{1,\dots,M\} \;\big|\; k_m = k \text{ and } k' \in \mathcal{W}_m\big\},$$

be the set of hyperedges containing the vertices from worker k' required by worker k to compute v_m, so that $\mathcal{R}_k = \big\{k' \in \{1,\dots,K\} \setminus \{k\} \;\big|\; \exists m \in \mathbb{E}_{(k,k')}\big\}$. Thus, the set of all the hyperedges that will carry out some communication to worker k is

$$\mathbb{E}_{\mathcal{R}_k} = \bigcup_{k' \in \mathcal{R}_k} \mathbb{E}_{(k,k')} = \big\{m \in \{1,\dots,M\} \;\big|\; k_m = k \text{ and } \mathcal{W}_m \neq \varnothing\big\}.$$

As a result, $\mathbb{E}_k$ and $\mathbb{E}_{\mathcal{R}_k}$ form a partition of $\overline{\mathbb{E}}_k$, where $\mathbb{E}_k$ corresponds to the set of weights v_m that can be computed locally on worker k (i.e. hyperedges inducing no communication), and $\mathbb{E}_{\mathcal{R}_k}$ 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

$$\mathbb{V}_{(k,k')} = \big\{n \in \mathbb{V}_{k'} \;\big|\; \exists m \in \mathbb{E}_{(k,k')} \text{ such that } n \in e_m\big\}$$

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 $\mathbb{V}_{\mathcal{W}_m} = \bigcup_{k' \in \mathcal{W}_m} \mathbb{V}_{(k_m,k')}$. For completeness, we also introduce $\mathbb{V}_{\overline{\mathcal{W}}_m} = \mathbb{V}_{k_m} \cup \mathbb{V}_{\mathcal{W}_m}$, the set of all the vertex indices necessary to perform computations associated with the m-th hyperedge.

For every $\boldsymbol{x} \in \mathbf{H}$, we denote by $\boldsymbol{x}_{(k,k')} = (x_n)_{n \in \mathbb{V}_{(k,k')}}$ the vertices communicated from worker k' to worker k through some hyperedge e_m such that $k_m = k$. For every $m \in \{1,\dots,M\} \setminus (\bigcup_{k=1}^{K} \mathbb{E}_k)$, let $\boldsymbol{x}_{\overline{\mathcal{W}}_m} = (x_n)_{n \in \mathbb{V}_{\overline{\mathcal{W}}_m}}$ 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 $k \in \{1,\dots,K\}$. For every $k' \in \mathcal{S}_k$, since $\mathbb{V}_{(k,k')}$ denotes the set of vertices that need to be communicated from worker k' to worker k, the (reciprocal) set $\mathbb{V}_{(k',k)} \subset \mathbb{V}_k$ corresponds to the set of vertices that will be sent back from worker k to k' in the distributed implementation of the adjoint operator $\boldsymbol{D}^*$. Similarly, with $\mathbb{E}_{(k,k')}$ the set of hyperedges containing vertices that will be communicated from worker k' to worker k, the set $\mathbb{E}_{(k',k)} \subset \{1,\dots,M\}$ 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 $\boldsymbol{D}$ over the workers. In practice, the input space $\mathbf{G}$ and the output space $\mathbf{H}$ are partitioned using the sets $(\overline{\mathbb{E}}_k)_{k \in \{1,\dots,K\}}$ and

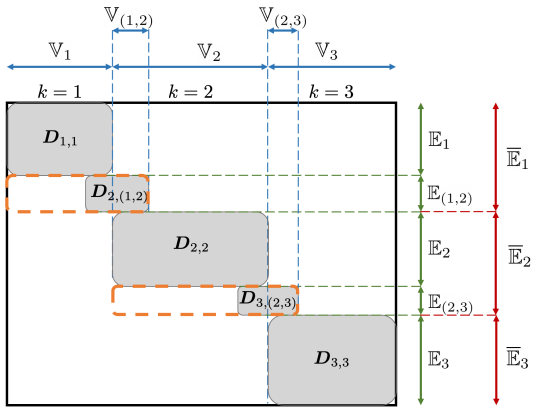


Figure 2: Distribution of vertices (horizontal lines) and hyperedges (vertical lines) of the hypergraph over $K = 3$ workers, for a block-sparse matrix $\boldsymbol{D}$ corresponding to Theorem 4.3.

$(\mathbb{V}_k)_{k\in\{1,\dots,K\}}$, respectively. For every $k \in \{1,\dots,K\}$, let $\mathbf{G}_k = \times_{m\in\overline{\mathbb{E}}_k} \mathsf{G}_m$ and $\mathbf{H}_k = \times_{n\in\mathbb{V}_k} \mathsf{H}_n$ such that $\mathbf{G} = \times_{1\leqslant k\leqslant K} \mathbf{G}_k$ and $\mathbf{H} = \times_{1\leqslant k\leqslant K} \mathbf{H}_k$.

For every $k \in \{1,\dots,K\}$, the subparts $\boldsymbol{v}_k$ of $\boldsymbol{D}\boldsymbol{x}$ that belong to $\mathbf{G}_k$ are stored on worker k. The subparts of $\boldsymbol{D}$ acting on $\mathbf{H}_k$ only are associated with vertices in $\mathbb{V}_k$. For every $m \in \mathbb{E}_k$, they will be denoted by $\boldsymbol{D}_{m,k} = (D_{m,n})_{n\in\mathbb{V}_k}$. These subparts of $\boldsymbol{D}$ are involved in purely local computations only. Similarly, for every $k' \in \mathcal{R}_k$, for every $m \in \mathbb{E}_{(k,k')}$, the subparts of $\boldsymbol{D}$ acting on vertices $\mathbb{V}_{(k,k')}$ will be denoted by $\boldsymbol{D}_{m,(k,k')} = (D_{m,n})_{n\in\mathbb{V}_k\cup\mathbb{V}_{(k,k')}}$. In addition, for every $m \in \mathbb{E}_{\mathcal{R}_k}$, we will denote by $\boldsymbol{D}_{m,\overline{\mathcal{W}}_m} = (D_{m,n})_{n\in\mathbb{V}_{\overline{\mathcal{W}}_m}}$ the subparts of $\boldsymbol{D}$ acting on vertices stored either on worker k_m or sent to worker k_m by all other workers $k' \in \mathcal{W}_m$. Hence

$$v_m = \begin{cases} \boldsymbol{D}_{m,k}\boldsymbol{x}_k, & \text{if } m \in \mathbb{E}_k, \\ \boldsymbol{D}_{m,\overline{\mathcal{W}}_m}\boldsymbol{x}_{\overline{\mathcal{W}}_m}, & \text{otherwise.} \end{cases} \tag{4.5}$$

As a result, the subparts of $\boldsymbol{D}$ which must be stored on worker k are the $\boldsymbol{D}_{m,k}$, for every $m \in \mathbb{E}_k$, and the $\boldsymbol{D}_{m,\overline{\mathcal{W}}_m}$, for every $m \in \mathbb{E}_{\mathcal{R}_k}$.

The notation given above is summarized in Table 1 and Table 2. A simple example for block-sparse matrices is also provided below, and illustrated in Figure 2.

Example 4.3 To illustrate some of the notation introduced above, consider the block-sparse matrix $\boldsymbol{D}$ shown in Figure 2, with $K = 3$ workers. For the vertices, $\mathbb{V}_1$ (resp. $\mathbb{V}_2$ and $\mathbb{V}_3$) contains the vertices handled on worker $k = 1$ (resp. $k = 2$ and $k = 3$). In addition, $\mathbb{V}_{(1,2)}$ (resp. $\mathbb{V}_{(2,3)}$) identifies the vertices of $\boldsymbol{x}_2$ (resp. $\boldsymbol{x}_3$) that will need to be communicated to worker $k = 1$ (resp. $k = 2$). For the hyperedges, $\overline{\mathbb{E}}_1$ (resp. $\overline{\mathbb{E}}_2$ and $\overline{\mathbb{E}}_3$) identifies the hyperedge weights stored on worker $k = 1$ (resp. $k = 2$ and $k = 3$). The set $\mathbb{E}_1$ (resp. $\mathbb{E}_2$ and $\mathbb{E}_3$) identifies the hyperedges fully contained in worker $k = 1$ (resp. $k = 2$ and $k = 3$). Hyperedges in $\mathbb{E}_{(1,2)}$ (resp. $\mathbb{E}_{(2,3)}$) contain 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 $\Big((\boldsymbol{D}_{m,k})_{m\in\mathbb{E}_k}\Big)_{1\leqslant k\leqslant 3}$, $(\boldsymbol{D}_{m,(1,2)})_{m\in\mathbb{E}_{(1,2)}}$ and $(\boldsymbol{D}_{m,(2,3)})_{m\in\mathbb{E}_{(2,3)}}$ are non-zero (e.g., convolution operator). Blocks $\Big((\boldsymbol{D}_{m,1})_{m\in\mathbb{E}_1}, (\boldsymbol{D}_{m,(1,2)})_{m\in\mathbb{E}_{(1,2)}}\Big)$ (resp. $\Big((\boldsymbol{D}_{m,2})_{m\in\mathbb{E}_2}, (\boldsymbol{D}_{m,(2,3)})_{m\in\mathbb{E}(2,3)}\Big)$ and $(\boldsymbol{D}_{m,3})_{m\in\mathbb{E}_3}$) 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 $\mathcal{R}_3=\varnothing$). Similarly, workers receiving vertex values from this worker is given by $\mathcal{S}_1=\varnothing$ (resp. $\mathcal{S}_2=\{1\}$ and $\mathcal{S}_3=\{2\}$). For every $m\in\overline{\mathbb{E}}_1$ (resp. $\overline{\mathbb{E}}_2$ and $\overline{\mathbb{E}}_3$), $k_m=1$ (resp. $k_m=2$ and $k_m=3$). For every $m\in\mathbb{E}_{(1,2)}$, $\mathcal{W}_m=\mathcal{R}_1=\mathcal{S}_2=\{2\}$, and for every $m\in\mathbb{E}_{(2,3)}$, $\mathcal{W}_m=\mathcal{R}_2=\{3\}$. Finally, for every $m\in\mathbb{E}_{\mathcal{R}_1}=\mathbb{E}_{(1,2)}$ (resp. $\mathbb{E}_{\mathcal{R}_2}=\mathbb{E}_{(2,3)}$), $\boldsymbol{D}_{m,\overline{\mathcal{W}}_m}=(D_{m,n})_{n\in\mathbb{V}_1\cup\mathbb{V}_{(1,2)}}$ (resp. $\boldsymbol{D}_{m,\overline{\mathcal{W}}_m}=(D_{m,n})_{n\in\mathbb{V}_2\cup\mathbb{V}_{(2,3)}}$). This last notation corresponds to the orange rectangles in Figure 1. In this example, $\mathbb{E}_{\mathcal{R}_3}=\varnothing$.

4.3 Separability assumptions

We can now give the assumptions that will be used to design an efficient distributed SGS algorithm. Precisely, we will consider distribution (1.2), where the functions are assumed to satisfy the following separability conditions.

Assumption 4.4

(i) $h\colon \mathbf{H}\to\mathbb{R}$ and $f\colon \mathbf{H}\to\,]-\infty,+\infty]$ are additively separable on the workers, i.e.

$$(\forall \boldsymbol{x}\in\mathbf{H})\quad h(\boldsymbol{x})=\sum_{k=1}^{K}h_k(\boldsymbol{x}_k),\ \text{and}\ f(\boldsymbol{x})=\sum_{k=1}^{K}f_k(\boldsymbol{x}_k), \tag{4.6}$$

where, for every $k\in\{1,\dots,K\}$, $h_k\colon \mathbf{H}_k\to\mathbb{R}$ and $f_k\colon \mathbf{H}_k\to\,]-\infty,+\infty]$.

(ii) $g\colon \mathbf{G}\to\,]-\infty,+\infty]$ is additively separable, i.e.

$$(\forall \boldsymbol{v}\in\mathbf{G})\quad g(\boldsymbol{v})=\sum_{m=1}^{M}g_m(v_m), \tag{4.7}$$

with, for every $m\in\{1,\dots,M\}$, $g_m\colon \mathsf{G}_m\to\,]-\infty,+\infty]$.

Further note that (2.7) and (2.8) can be written

$$(\forall(\boldsymbol{v},\boldsymbol{u})\in\mathbf{G}^2)\quad \phi_\alpha(\boldsymbol{v},\boldsymbol{u})=\sum_{m=1}^{M}\phi_{m,\alpha}(v_m,u_m)=\sum_{m=1}^{M}\frac{1}{2\alpha^2}\|v_m-u_m\|^2, \tag{4.8}$$

$$\psi_\beta(\boldsymbol{u})=\sum_{m=1}^{M}\psi_{m,\beta}(u_m)=\sum_{m=1}^{M}\frac{1}{2\beta^2}\|u_m\|^2. \tag{4.9}$$

The above assumptions can be rewritten to highlight the separability over the workers. Indeed, there exists a permutation $\varrho\colon \mathbf{G} \to \mathbf{G}$ such that

$$(\forall \boldsymbol{x} \in \mathbf{H}) \quad \boldsymbol{D}\boldsymbol{x} = \Big((D_{m,n})_{1 \leqslant n \leqslant N} \, \boldsymbol{x} \Big)_{1 \leqslant m \leqslant M} = \varrho \left(\begin{pmatrix} (\boldsymbol{D}_{m,k})_{m \in \mathbb{E}_k} \, \boldsymbol{x}_k \\ \Big(\boldsymbol{D}_{m,\overline{\mathcal{W}}_m} \boldsymbol{x}_{\overline{\mathcal{W}}_m} \Big)_{m \in \mathbb{E}_{\mathcal{R}_k}} \end{pmatrix}_{1 \leqslant k \leqslant K} \right). \tag{4.10}$$

Reorganizing the vertex indices emphasizes the distinction between local computations and those for which communications are required. For every $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbf{G}^2$ and $\boldsymbol{x} \in \mathbf{H}$, (4.7), (4.8) and (4.9) can thus be rewritten as

$$g(\boldsymbol{v}) = \sum_{k=1}^{K} \Big(\sum_{m \in \overline{\mathbb{E}}_k} g_m \left(v_m \right) \Big), \tag{4.11}$$

$$\phi_\alpha(\boldsymbol{D}\boldsymbol{x}, \boldsymbol{u}) = \sum_{k=1}^{K} \Big(\sum_{m \in \overline{\mathbb{E}}_k} \phi_{m,\alpha} \left([\boldsymbol{D}\boldsymbol{x}]_m, u_m \right) \Big), \tag{4.12}$$

$$= \sum_{k=1}^{K} \Big(\sum_{m \in \mathbb{E}_k} \phi_{m,\alpha} \left(\boldsymbol{D}_{m,k} \boldsymbol{x}_k, u_m \right) + \sum_{m \in \mathbb{E}_{\mathcal{R}_k}} \phi_{m,\alpha} \left(\boldsymbol{D}_{m,\overline{\mathcal{W}}_m} \boldsymbol{x}_{\overline{\mathcal{W}}_m}, u_m \right) \Big), \tag{4.13}$$

$$\psi_\beta(\boldsymbol{u}) = \sum_{k=1}^{K} \Big(\sum_{m \in \overline{\mathbb{E}}_k} \psi_{m,\beta} \left(u_m \right) \Big), \tag{4.14}$$

where $[\cdot]_m$ denotes the m-th element of its argument.

5 Distributed Split 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. The term *block-coordinate* is to be interpreted as in the optimization literature, where the variables from (1.2) 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* (1.2) *satisfying Theorems 2.2 and 4.4. Assume that the operator $\boldsymbol{D}$ follows Theorem 4.1, and is split over workers $\{1, \dots, K\}$ such that, for every $k \in \{1, \dots, K\}$, $(\boldsymbol{D}_{m,k})_{m \in \overline{\mathbb{E}}_k}$ is stored on worker k. For every $k \in \{1, \dots, K\}$, let $\boldsymbol{x}_k^{(0)} \in \mathbf{H}_k$, $\boldsymbol{z}_k^{(0)} \in \mathbf{G}_k$, and $\boldsymbol{u}_k^{(0)} \in \mathbf{G}_k$. Let $(\boldsymbol{x}^{(t)}, \boldsymbol{z}^{(t)}, \boldsymbol{u}^{(t)})_{1 \leqslant t \leqslant T}$ be samples generated by Algorithm 3, where $\gamma \in]0, (\lambda_h + \|\boldsymbol{D}\|^2/\alpha^2)^{-1}[$ and, for every $k \in \{1, \dots, K\}$, $(\boldsymbol{w}_k^{(t)})_{1 \leqslant t \leqslant T}$ is a sequence of i.i.d. standard Gaussian random variables in*

$\mathbf{H}_k$. *In addition, the transition kernel* $\mathcal{K}_{\boldsymbol{z}_k,\alpha}$ *has the following invariant distribution*

$$\pi_{k,\alpha}\left(\boldsymbol{z}_k \mid \boldsymbol{v}_k, \boldsymbol{u}_k\right) \propto \exp\left(- \sum_{m \in \overline{\mathbb{E}}_k} \left(g_m(z_m) + \phi_{m,\alpha}(v_m, z_m - u_m)\right) \right) \tag{5.1}$$

where, for every $\boldsymbol{x} \in \mathbf{H}$, $\boldsymbol{v}_k = (v_m)_{m \in \overline{\mathbb{E}}_k} = \varrho \begin{pmatrix} (\boldsymbol{D}_{m,k}\boldsymbol{x}_k)_{m \in \mathbb{E}_k} \\ (\boldsymbol{D}_{m,\overline{\mathcal{W}}_m}\boldsymbol{x}_{\overline{\mathcal{W}}_m})_{m \in \mathbb{E}_{\mathcal{R}_k}} \end{pmatrix}$.
Then, Algorithm 3 is equivalent to Algorithm 2.

All the computations described in Algorithm 3 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 $\boldsymbol{D}$ and its adjoint $\boldsymbol{D}^*$ 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 $\boldsymbol{D}$ are received by worker k from those of $\mathcal{R}_k$; communications necessary to computations involving $\boldsymbol{D}^*$ are received by worker k from those of $\mathcal{S}_k$ (see Section 5.2).

Proof. Theorem 2.2 ensures that Theorem 2.1 is verified (see (2.6)), and that the SGS algorithm can be instantiated as Algorithm 2.

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 $\boldsymbol{v} = \boldsymbol{D}\boldsymbol{x}$, the conditional distributions (2.2)-(2.3) can be rewritten as

$$\pi_{\alpha}\left(\boldsymbol{z} \mid \boldsymbol{v}, \boldsymbol{u}\right) = \prod_{k=1}^{K} \pi_{k,\alpha}\left(\boldsymbol{z}_k \mid \boldsymbol{v}_k, \boldsymbol{u}_k\right), \tag{5.2}$$

$$\pi_{(\alpha,\beta)}\left(\boldsymbol{u} \mid \boldsymbol{v}, \boldsymbol{z}\right) = \prod_{k=1}^{K} \pi_{k,(\alpha,\beta)}\left(\boldsymbol{u}_k \mid \boldsymbol{v}_k, \boldsymbol{z}_k\right), \tag{5.3}$$

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

$$\pi_{k,\alpha}\left(\boldsymbol{z}_k \mid \boldsymbol{v}_k, \boldsymbol{u}_k\right) \propto \exp\left(- \sum_{m \in \overline{\mathbb{E}}_k} \left(g_m(z_m) + \phi_{m,\alpha}(v_m, z_m - u_m)\right) \right), \tag{5.4}$$

$$\begin{aligned} \pi_{k,(\alpha,\beta)}\left(\boldsymbol{u}_k \mid \boldsymbol{v}_k, \boldsymbol{z}_k\right) &\propto \exp\left(- \sum_{m \in \overline{\mathbb{E}}_k} \left(\phi_{m,\alpha}(v_m, z_m - u_m) - \psi_{m,\beta}(u_m)\right) \right), \\ &= \mathcal{N}\left(\frac{\beta^2}{\alpha^2+\beta^2}(\boldsymbol{z}_k - \boldsymbol{v}_k), \frac{\alpha^2+\beta^2}{\alpha^2\beta^2}\mathbf{I} \right), \end{aligned} \tag{5.5}$$

for $\boldsymbol{v}_k = (v_m)_{m \in \overline{\mathbb{E}}_k} = ([\boldsymbol{D}\boldsymbol{x}]_m)_{m \in \overline{\mathbb{E}}_k}$. Hence, for every iteration $t \in \{0, \ldots, T\}$ of the Algorithm 1, and for every worker $k \in \{1, \ldots, K\}$, the sampling of $\boldsymbol{z}_k^{(t+1)}$ and $\boldsymbol{u}_k^{(t+1)}$ only requires partial

Algorithm 3: Proposed distributed SGS (distributed version of Algorithm 2).

for $k = 1$ **to** K **do**
  for $k' \in \mathcal{S}_k$ **do**
    Send $(x_n^{(0)})_{n\in\mathbb{V}_{(k',k)}}$ to worker k';
  for $k' \in \mathcal{R}_k$ **do**
    Receive $(x_n^{(0)})_{n\in\mathbb{V}_{(k,k')}}$ from worker k';
  $\boldsymbol{v}_k^{(0)} = \varrho\begin{pmatrix}(\boldsymbol{D}_{m,k}\boldsymbol{x}_k^{(0)})_{m\in\mathbb{E}_k}\\ (\boldsymbol{D}_{m,\overline{\mathcal{W}}_m}\boldsymbol{x}_{\overline{\mathcal{W}}_m}^{(0)})_{m\in\mathbb{E}_{\mathcal{R}_k}}\end{pmatrix}$;
for $t = 0$ **to** T **do**
  for $k = 1$ **to** K **do**
    $(d_m^{(t)})_{m\in\overline{\mathbb{E}}_k} = \left(\phi'_{m,\alpha}(\cdot, z_m^{(t)} - u_m^{(t)})(v_m^{(t)})\right)_{m\in\overline{\mathbb{E}}_k}$;
    for $k' \in \mathcal{R}_k$ **do**
      $\widetilde{\boldsymbol{d}}_{(k',k)}^{(t)} = \sum_{m\in\mathbb{E}_{(k',k)}} \boldsymbol{D}_{m,k'}^* d_m^{(t)}$;
      Send $\widetilde{\boldsymbol{d}}_{(k',k)}^{(t)}$ to worker k';
    for $k' \in \mathcal{S}_k$ **do**
      Receive $\widetilde{\boldsymbol{d}}_{(k,k')}^{(t)}$ from worker k';
    $\boldsymbol{\delta}_k^{(t)} = \sum_{m\in\overline{\mathbb{E}}_k} \boldsymbol{D}_{m,k}^* d_m^{(t)} + \sum_{k'\in\mathcal{S}_k} \widetilde{\boldsymbol{d}}_{(k,k')}^{(t)}$;
    $\boldsymbol{x}_k^{(t+1)} = \operatorname{prox}_{\gamma f_k}\left(\boldsymbol{x}_k^{(t)} - \gamma\nabla h_k(\boldsymbol{x}_k^{(t)}) - \gamma\boldsymbol{\delta}_k^{(t)} + \sqrt{2\gamma}\boldsymbol{w}_k^{(t)}\right)$;
    for $k' \in \mathcal{S}_k$ **do**
      Send $(x_n^{(t+1)})_{n\in\mathbb{V}_{(k',k)}}$ to worker k';
    for $k' \in \mathcal{R}_k$ **do**
      Receive $(x_n^{(t+1)})_{n\in\mathbb{V}_{(k,k')}}$ from worker k';
    $\boldsymbol{v}_k^{(t+1)} = \varrho\begin{pmatrix}(\boldsymbol{D}_{m,k}\boldsymbol{x}_k^{(t+1)})_{m\in\mathbb{E}_k}\\ (\boldsymbol{D}_{m,\overline{\mathcal{W}}_m}\boldsymbol{x}_{\overline{\mathcal{W}}_m}^{(t+1)})_{m\in\mathbb{E}_{\mathcal{R}_k}}\end{pmatrix}$;
    $\boldsymbol{z}_k^{(t+1)} \sim \pi_{k,\alpha}\left(\boldsymbol{z}_k \mid \boldsymbol{v}_k^{(t+1)}, \boldsymbol{u}_k^{(t)}\right)$;
    $\boldsymbol{u}_k^{(t+1)} \sim \mathcal{N}\left(\frac{\beta^2}{\alpha^2+\beta^2}(\boldsymbol{z}_k^{(t+1)} - \boldsymbol{v}_k^{(t+1)}), \frac{\alpha^2+\beta^2}{\alpha^2\beta^2}\mathbf{I}\right)$;

information from $\left(\boldsymbol{v}_k^{(t+1)}, \boldsymbol{u}_k^{(t)}\right)$ and $\left(\boldsymbol{v}_k^{(t+1)}, \boldsymbol{z}_k^{(t+1)}\right)$, respectively. Algorithm 2 is thus equivalent to

$$
\begin{aligned}
&\text{for } t = 0, 1, \dots, T \\
&\quad \boldsymbol{x}^{(t+1)} = \operatorname{prox}_{\gamma f}\Big(\boldsymbol{x}^{(t)} - \gamma \nabla h(\boldsymbol{x}^{(t)}) - \gamma \boldsymbol{D}^* \Big(\nabla \phi_\alpha(\cdot, \boldsymbol{z}^{(t)} - \boldsymbol{u}^{(t)})((v_m^{(t)})_{m \in \overline{\mathbb{E}}_k})\Big) \\
&\qquad\qquad + \sqrt{2\gamma}\, \boldsymbol{w}^{(t)}\Big) \\
&\quad \text{for } k = 1, \dots, K \\
&\quad\quad \boldsymbol{v}_k^{(t+1)} = ([\boldsymbol{D}\boldsymbol{x}^{(t+1)}]_m)_{m \in \overline{\mathbb{E}}_k}, \\
&\quad\quad \boldsymbol{z}_k^{(t+1)} \sim \pi_{k,\alpha}\left(\boldsymbol{z}_k \mid \boldsymbol{v}_k^{(t+1)}, \boldsymbol{u}_k^{(t)}\right), \\
&\quad\quad \boldsymbol{u}_k^{(t+1)} \sim \mathcal{N}\Big(\tfrac{\beta^2}{\alpha^2+\beta^2}(\boldsymbol{z}_k^{(t+1)} - \boldsymbol{v}_k^{(t+1)}), \tfrac{\alpha^2+\beta^2}{\alpha^2\beta^2}\mathbf{I}\Big).
\end{aligned}
\tag{5.6}
$$

It remains to show that, for every $t \in \{0, \dots, T\}$, the computation of $\boldsymbol{x}^{(t+1)}$ can be parallelized over $k \in \{1, \dots, K\}$. Since $(\mathbb{V}_k)_{1 \leqslant k \leqslant K}$ is a partition of $\{1, \dots, N\}$, then, for every $k \in \{1, \dots, K\}$, $\boldsymbol{x}_k^{(t+1)}$ 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 $\boldsymbol{x}$ is required to compute $(\boldsymbol{x}_k^{(t+1)})_{1 \leqslant k \leqslant K}$. Using the chain rule on $\boldsymbol{x} \in \mathbf{H} \mapsto (\phi_\alpha(\cdot, \boldsymbol{z} - \boldsymbol{u}) \circ \boldsymbol{D})(\boldsymbol{x})$, for $(\boldsymbol{z}, \boldsymbol{u}) \in \mathbf{G}^2$ fixed, we obtain, for every $\boldsymbol{x} \in \mathbf{H}$,

$$
\nabla_{\boldsymbol{x}} \phi_\alpha(\boldsymbol{D}\boldsymbol{x}, \boldsymbol{z} - \boldsymbol{u}) = \nabla_{\boldsymbol{x}}(\phi_\alpha(\cdot, \boldsymbol{z} - \boldsymbol{u}) \circ \boldsymbol{D})(\boldsymbol{x}) = \boldsymbol{D}^* \nabla \phi_\alpha(\cdot, \boldsymbol{z} - \boldsymbol{u})(\boldsymbol{D}\boldsymbol{x}). \tag{5.7}
$$

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

$$
\nabla_{\boldsymbol{v}} \phi_\alpha(\boldsymbol{v}, \boldsymbol{z}) = \varrho\left(\left(\begin{pmatrix} \left(\phi'_{m,\alpha}(v_m, z_m)\right)_{m \in \mathbb{E}_k} \\ \left(\phi'_{m,\alpha}(v_m, z_m)\right)_{m \in \mathbb{E}_{\mathcal{R}_k}} \end{pmatrix}\right)_{1 \leqslant k \leqslant K}\right), \tag{5.8}
$$

where ϱ is the permutation operator defined in (4.10). For every $k \in \{1, \dots, K\}$, every $\boldsymbol{x} \in \mathbf{H}$ and $(\boldsymbol{z}, \boldsymbol{u}) \in \mathbf{G}^2$, let $(d_m)_{m \in \overline{\mathbb{E}}_k}$ be defined as

$$
d_m := \phi'_{m,\alpha}(\cdot, z_m - u_m)(v_m) = \begin{cases} \phi'_{m,\alpha}(\cdot, z_m - u_m)(\boldsymbol{D}_{m,k}\boldsymbol{x}_k), & \text{if } m \in \mathbb{E}_k, \\ \phi'_{m,\alpha}(\cdot, z_m - u_m)(\boldsymbol{D}_{m,\overline{\mathcal{W}}_m}\boldsymbol{x}_{\overline{\mathcal{W}}_m}), & \text{if } m \in \mathbb{E}_{\mathcal{R}_k}. \end{cases} \tag{5.9}
$$

Finally,

$$
\nabla_{\boldsymbol{x}} \phi_\alpha(\boldsymbol{D}\boldsymbol{x}, \boldsymbol{z} - \boldsymbol{u}) = \boldsymbol{D}^* \varrho\left(\left((d_m)_{m \in \overline{\mathbb{E}}_k}\right)_{1 \leqslant k \leqslant K}\right). \tag{5.10}
$$

To extract the k-th block from this gradient, (5.10) can be decomposed as follows:

$$
\begin{aligned}
\boldsymbol{\delta}_k &:= [\nabla_{\boldsymbol{x}} \phi_\alpha(\boldsymbol{D}\boldsymbol{x}, \boldsymbol{z} - \boldsymbol{u})]_k \\
&= \sum_{m \in \overline{\mathbb{E}}_k} \boldsymbol{D}^*_{m,k} d_m + \sum_{k' \in \mathcal{S}_k} \sum_{m \in \mathbb{E}_{(k,k')}} \boldsymbol{D}^*_{m,k} d_m.
\end{aligned}
\tag{5.11}
$$

Figure 4 illustrates this for the simple example described in Figure 2.

The second term in (5.11) gathers information received from the set of neighbours $\mathcal{S}_k$ of worker k that is necessary to computations involving $\boldsymbol{D}^*$. Then, using the notation

$$(\forall k' \in \mathcal{S}_k) \quad \widetilde{\boldsymbol{d}}_{(k,k')} = \sum_{m \in \mathbb{E}_{(k,k')}} \boldsymbol{D}^*_{m,k} d_m, \tag{5.12}$$

we obtain

$$\boldsymbol{\delta}_k = \sum_{m \in \overline{\mathbb{E}}_k} \boldsymbol{D}^*_{m,k} d_m + \sum_{k' \in \mathcal{S}_k} \widetilde{\boldsymbol{d}}_{(k,k')}, \tag{5.13}$$

where, for every $k' \in \mathcal{S}_k$, $\widetilde{\boldsymbol{d}}_{(k,k')}$ is computed on worker k' and communicated to worker k to form $\boldsymbol{\delta}_k$. Therefore the computations performed on a given worker $k \in \{1, \dots, K\}$ include the computation of *local* gradients $(\boldsymbol{D}^*_{m,k} d_m)_{m \in \overline{\mathbb{E}}_k}$ as well as the gradients to be sent to neighbours, i.e., $(\widetilde{\boldsymbol{d}}_{(k',k)})_{k' \in \mathcal{R}_k}$. Note that the role of the sets $\mathcal{R}_k$ and $\mathcal{S}_k$ is exchanged when the adjoint operator $\boldsymbol{D}^*$ is considered.

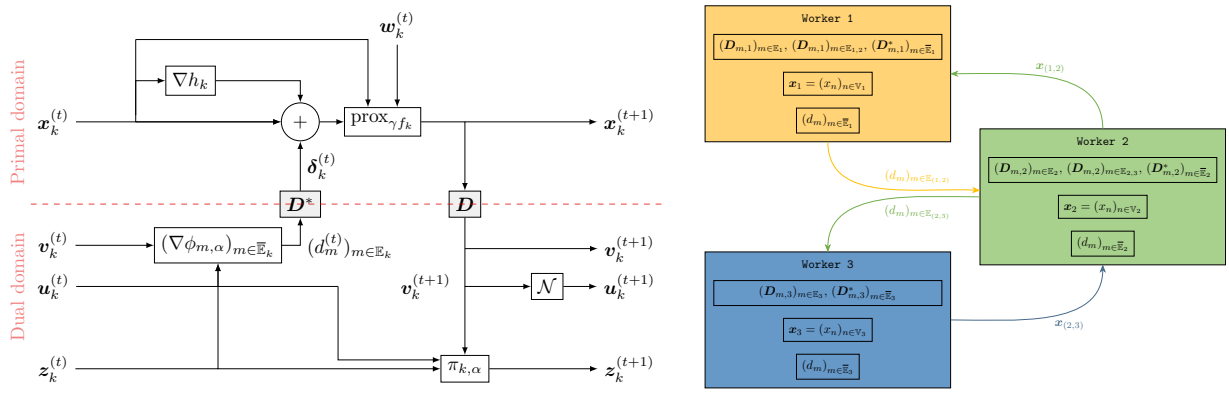
Using the above notation, and using [1, Prop. 24.11] on the additively separable functions h and f, leads to the conclusion that Algorithm 3 is equivalent to (5.6), and therefore to Algorithm 2. □

5.2 Illustration of operator computation

In this section, we give an example to explain the communications involved in Algorithm 3. Figure 3 describes one iteration of Algorithm 3. Figure 4 describes the application of the forward $\boldsymbol{D}$ and backward $\boldsymbol{D}^*$ operators.

Figure 3 (a) illustrates one iteration of the proposed distributed sampling Algorithm 3 implemented on an SPMD architecture, with local computations conducted on a worker $k \in \{1, \dots, K\}$. Theorem 4.4 ensures that most of the related operations can be conducted independently on each worker. Applying the forward operator $\boldsymbol{D}$ and its adjoint $\boldsymbol{D}^*$ requires communications illustrated in Figure 3 (b).

Figure 4(a) illustrates the computation of $\boldsymbol{D}\boldsymbol{x}$, for $\boldsymbol{x} \in \mathbf{H}$ with the block-sparse matrix $\boldsymbol{D}$ of Example 4.3, see also Figure 1. Figure 4(b) illustrates the computation of $\boldsymbol{D}^*\boldsymbol{d}$, for $\boldsymbol{d} \in \mathbf{G}$ with some adjoint symmetry.
Subparts $\mathbb{V}_{(1,2)}$ of $\boldsymbol{x}_2$ and $\mathbb{V}_{(2,3)}$ of $\boldsymbol{x}_3$ need to be communicated to workers 1 and 2, respectively, to compute $\boldsymbol{D}\boldsymbol{x}$. In Figure 4(a), for worker $k = 1$, the quantity $(\boldsymbol{D}_{m,\overline{\mathcal{W}}_m} \boldsymbol{x}_{\overline{\mathcal{W}}_m})_{m \in \mathbb{E}_{\mathcal{R}_1}}$ can be computed by multiplying the subpart of $\boldsymbol{D}$ corresponding to $(\boldsymbol{D}_{m,\overline{\mathcal{W}}_m})_{m \in \mathbb{E}_{\mathcal{R}_1}}$ (see Figure 4 (a), horizontal rectangle in continuous yellow lines) with $(\boldsymbol{x}_{\overline{\mathcal{W}}_m})_{m \in \mathbb{E}_{\mathcal{R}_1}} = (\boldsymbol{x}_1, \boldsymbol{x}_{(1,2)})$ (vertical rectangle in continuous yellow lines). This quantity is computed on worker 1, once worker 2 has communicated $\boldsymbol{x}_{(1,2)}$ to worker $k = 1$. Similarly, for worker $k = 2$, the quantity $(\boldsymbol{D}_{m,\overline{\mathcal{W}}_m} \boldsymbol{x}_{\overline{\mathcal{W}}_m})_{m \in \mathbb{E}_{\mathcal{R}_2}}$ can be



(a) Iteration k of the proposed sampler

(b) Communications between workers

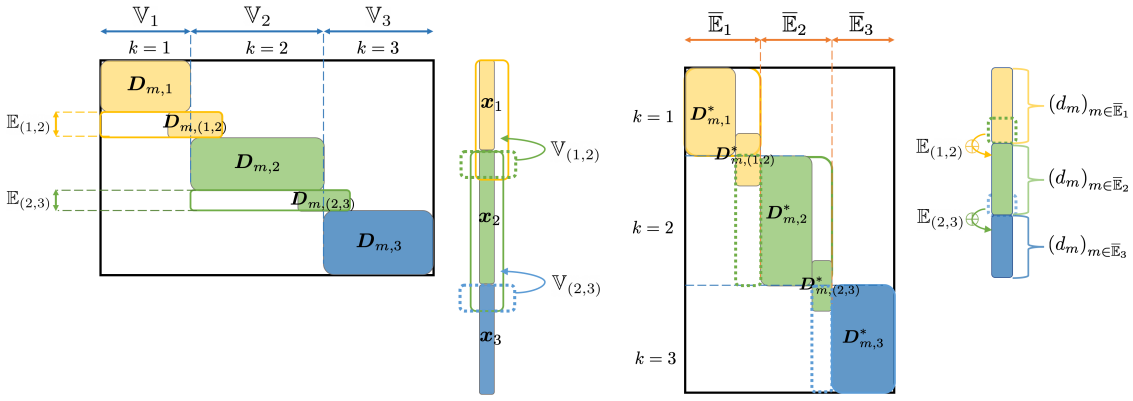
Figure 3: (a) Operations required by one iteration $k \in \mathbb{N}^*$ of Algorithm 3, a distributed version of Algorithm 2 using an SPMD architecture. Shaded rectangles correspond to operations for which communications are required. (b) Communications required by the distributed implementation of Algorithm 3, associated with the operator $\boldsymbol{D}$ illustrated in Figure 4. Local variables are displayed in the boxes, and variables exchanged between workers are represented with colored arrows.

computed by multiplying the subpart of $\boldsymbol{D}$ corresponding to $(\boldsymbol{D}_{m,\overline{\mathcal{W}}_m})_{m\in\mathbb{E}_{\mathcal{R}_2}}$ (horizontal rectangle in continuous green lines) with $(\boldsymbol{x}_{\overline{\mathcal{W}}_m})_{m\in\mathbb{E}_{\mathcal{R}_2}} = (\boldsymbol{x}_2, \boldsymbol{x}_{(2,3)})$ (vertical rectangle in continuous green lines). This quantity is computed on worker 2, once worker 3 has communicated $\boldsymbol{x}_{(2,3)}$ to worker $k = 2$.

To compute $\boldsymbol{D}^*\boldsymbol{d}$, communications are performed after computing subparts of $\boldsymbol{D}^*\boldsymbol{d}$. In Figure 4(b), for worker $k = 1$, the quantity $[\boldsymbol{D}^*\boldsymbol{d}]_1$ can be computed by multiplying the subpart of $\boldsymbol{D}^*$ corresponding to $(\boldsymbol{D}^*_{m,1})_{m\in\overline{\mathbb{E}}_1}$ (continuous-line yellow rectangle, top left) with $(d_m)_{m\in\overline{\mathbb{E}}_1}$. For the computation of $[\boldsymbol{D}^*\boldsymbol{d}]_1$, no communication with other workers is needed in this example. For worker $k = 2$, $[\boldsymbol{D}^*\boldsymbol{d}]_2$ needs to be decomposed between parts of $\boldsymbol{D}^*$ stored on worker $k = 2$, and parts of $\boldsymbol{D}$ that are stored on other workers, i.e., $k = 1$ for this example. For the parts of $\boldsymbol{D}^*$ stored on worker $k = 2$, the subpart of $\boldsymbol{D}^*$ corresponding to $(\boldsymbol{D}^*_{m,2})_{m\in\overline{\mathbb{E}}_2}$ (continuous-line green rectangle) needs to be multiplied with $(d_m)_{m\in\overline{\mathbb{E}}_2}$. For the parts of $\boldsymbol{D}^*$ stored on $k = 1$, the subpart of $\boldsymbol{D}^*$ corresponding to $(\boldsymbol{D}^*_{m,2})_{m\in\mathbb{E}_{(1,2)}})$ (dotted-line green rectangle) needs to be multiplied with $(d_m)_{m\in\mathbb{E}_{(1,2)}}$ (dotted-line green rectangle on vector $\boldsymbol{d}$). 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 $[\boldsymbol{D}^*\boldsymbol{d}]_3$ is similar to the one taken for worker $k = 2$.

5.3 Distributed SPMD architecture

The distributed block-coordinate sampler described in the previous section can benefit from an implementation on a 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



(a) Computation of $\boldsymbol{D}\boldsymbol{x}$ (b) Computation of $\boldsymbol{D}^*\boldsymbol{d}$

Figure 4: Illustration of the computation of (a) $\boldsymbol{D}\boldsymbol{x}$ for $\boldsymbol{x} \in \mathbf{H}$, and (b) $\boldsymbol{D}^*\boldsymbol{d}$ for $\boldsymbol{d} \in \mathbf{G}$. $\boldsymbol{D}$ corresponds to the block-sparse matrix from Figure 2, and $\boldsymbol{D}^*$ to its adjoint. The yellow, green and blue colours identify how $\boldsymbol{D}$, $\boldsymbol{D}^*$, $\boldsymbol{x}$ and $\boldsymbol{d}$ are distributed over the workers $k=1$, $k=2$, and $k=3$, respectively. Communications are illustrated by coloured arrows over subparts of $\boldsymbol{x}$ and $\boldsymbol{d}$, respectively. Continuous coloured lines on $\boldsymbol{x}$ and $\boldsymbol{d}$ identify vertices and hyperedge weights required to perform local computations. The corresponding blocks in $\boldsymbol{D}$ and $\boldsymbol{D}^*$ 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 $\boldsymbol{D}$ and $\boldsymbol{D}^*$ highlighted accordingly. (a) Vertices are on columns, and hyperedges on rows. (b) Vertices are on rows, and hyperedges on columns. The "plus" symbol emphasizes that hyperedge weights are aggregated upon reception.

and observations [15]. This enables the hypergraph structure of $\boldsymbol{D}$ 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 are compatible with an SPMD architecture. The separability Assumptions 4.4-(i) and (ii) ensure that K conditionally independent blocks can be formed for variables $\boldsymbol{v} \in \mathbf{G}$ in the dual domain. The separability of g further implies that the evaluation of its proximity operator is easy to parallelize. Combining the definition of ϕ_α and ψ_β (Theorem 2.2-(i)) with the structure of $\boldsymbol{D}$ finally enables Algorithm 2 to be reformulated using blocks of parameters, each stored on (only) one of the K workers. Using an SPMD architecture for Algorithm 3 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.

(ii) Parallelization flexibility: an SPMD architecture can be readily used to address (1.1) under Theorem 4.4, whereas a client-server architecture cannot (no conditionally independent blocks of variables as in [28])

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

(iv) Communications and data locality: 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 3.3 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 (1.1). 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 $(\boldsymbol{D}_i)_{1\leqslant i\leqslant I}$. The number of workers it can accommodate is also restricted to the number of conditionally independent blocks of variables, as in [37, 28] (see Section 3.1).

6 Application to supervised image deconvolution

To show the performance of the proposed distributed SGS, we use it to solve an image deconvolution problem. Image deconvolution is an inverse problem that consists in inferring an unknown variable $\overline{\boldsymbol{x}}$ from observations $\boldsymbol{y}$. Observations and parameters are typically related by a model of the form

$$\boldsymbol{y} = \mathcal{D}(\boldsymbol{A}\overline{\boldsymbol{x}}), \tag{6.1}$$

where the linear operator $\boldsymbol{A}$ models the acquisition process, and $\mathcal{D}$ models random perturbations – referred to as noise – damaging the clean data $\boldsymbol{A}\overline{\boldsymbol{x}}$. Bayesian inference relies on the posterior distribution of the random variable $\boldsymbol{x}$ to estimate the true value $\overline{\boldsymbol{x}}$. The posterior distribution, often of the form (1.1), combines information from the likelihood – related to the observations $\boldsymbol{y}$ – 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 [30], or a wavelet basis [23]. Prior information can also encompass constraints based on the physics of the data acquisition process, such as nonnegativity for intensity images [5, 32], or polarization constraints [3].

We consider 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 [33] for an inpainting problem (*i.e.*, with $\boldsymbol{A}$ 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 $C = 2$ composite terms. The notation used below corresponds to the one introduced in Appendix Afor distributions with multiple composite terms.

6.1 Problem statement

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

$$(\forall m \in \{1, \dots, M\}) \quad y_m \sim \mathcal{P}([\boldsymbol{D}_1 \overline{\boldsymbol{x}}]_m), \tag{6.2}$$

where $\boldsymbol{D}_1 \in \mathbb{R}^{M \times N}$ is a 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 μ. In this context, $\overline{N} = N$ and, for every $n \in \{1, \dots, N\}$, $\mathsf{H}_n = \mathbb{R}$. Similarly, $\overline{M} = M$ and, for every $m \in \{1, \dots, M\}$, $\mathsf{G}_m = \mathbb{R}$.

We propose to solve this problem with a hybrid prior combining a non-negativity constraint and a TV regularization [30]. Such prior has for instance been considered in [17, 35]. The resulting posterior distribution is given by

$$\pi(\boldsymbol{x}) \propto \exp\left(-f(\boldsymbol{x}) - g_1(\boldsymbol{D}_1 \boldsymbol{x}) - g_2(\boldsymbol{D}_2 \boldsymbol{x})\right), \tag{6.3}$$

where $g_1 \colon \mathbb{R}^M \to]-\infty, +\infty] \colon (z_m)_{1 \leqslant m \leqslant M} \mapsto \sum_{m=1}^M g_{1,m}(z_m)$ is the data-fidelity term, with

$$(\forall m \in \{1, \dots, M\}) \quad g_{1,m}(z_m) = -y_m \log(z_m) + z_m, \tag{6.4}$$

due to the Poisson distribution, $f = \iota_{[0,+\infty[^N} \colon \mathbb{R}^N \to]-\infty, +\infty]$ is the indicator function of the positive orthant, and $g_2 \circ \boldsymbol{D}_2$ models the discrete isotropic TV [30]. Precisely, $\boldsymbol{D}_2 \colon \mathbb{R}^N \to \mathbb{R}^{2N}$ is the concatenation of the vertical and horizontal discrete gradients, and $g_2 : \mathbb{R}^{2N} \to]-\infty, +\infty]$ is the $\ell_{2,1}$-norm

$$\begin{aligned} &(\forall \boldsymbol{z} = (\boldsymbol{z}_n)_{1 \leqslant n \leqslant N} \in \mathbb{R}^{2 \times N}) \quad g_2(\boldsymbol{z}) = \sum_{n=1}^N g_{2,n}(\boldsymbol{z}_n), \\ &(\forall n \in \{1, \dots, N\})(\forall \boldsymbol{z}_n \in \mathbb{R}^2) \quad g_{2,n}(\boldsymbol{z}_n) = \kappa \|\boldsymbol{z}_n\|_2, \quad \text{where } \kappa > 0. \end{aligned} \tag{6.5}$$

As a first step towards a distributed implementation, the AXDA approach (see Appendix Afor details) is used to approximate $\pi(\boldsymbol{x})$ by

$$\pi_{\boldsymbol{\alpha},\boldsymbol{\beta}}(\boldsymbol{x}, (\boldsymbol{z}_c, \boldsymbol{u}_c)_{1 \leqslant c \leqslant 2}) \propto \exp\left(-f(\boldsymbol{x}) - \sum_{c=1}^2 \left(g_c(\boldsymbol{z}_c) + \phi_{c,\alpha_c}(\boldsymbol{D}_c \boldsymbol{x}, \boldsymbol{z}_c - \boldsymbol{u}_c) + \psi_{c,\beta_c}(\boldsymbol{u}_c)\right)\right), \tag{6.6}$$

where $(\alpha_1, \alpha_2, \beta_1, \beta_2) \in]0, +\infty[^4$, and the functions $(\phi_{c,\alpha_c}, \psi_{c,\beta_c})_{1 \leqslant c \leqslant 2}$ are defined in (2.6).

The directed acyclic graph reported in Figure 5 summarizes the structure of the approximate posterior distribution (6.6), highlighting dependencies between the variables. Note that $(\boldsymbol{z}_1, \boldsymbol{u}_1)$ and $(\boldsymbol{z}_2, \boldsymbol{u}_2)$ are the only conditionally independent blocks of variables.

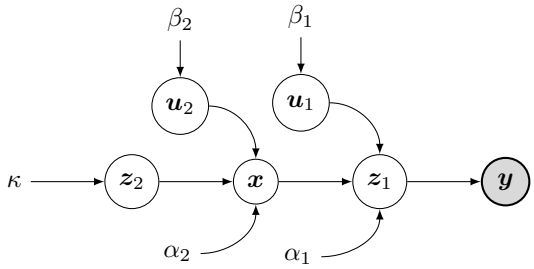


Figure 5: Directed acyclic graph describing the factorization of the approximate posterior distribution (6.6). Circled variables correspond to random variables. Other variables are fixed *a priori*.

6.2 Proposed SPMD implementation

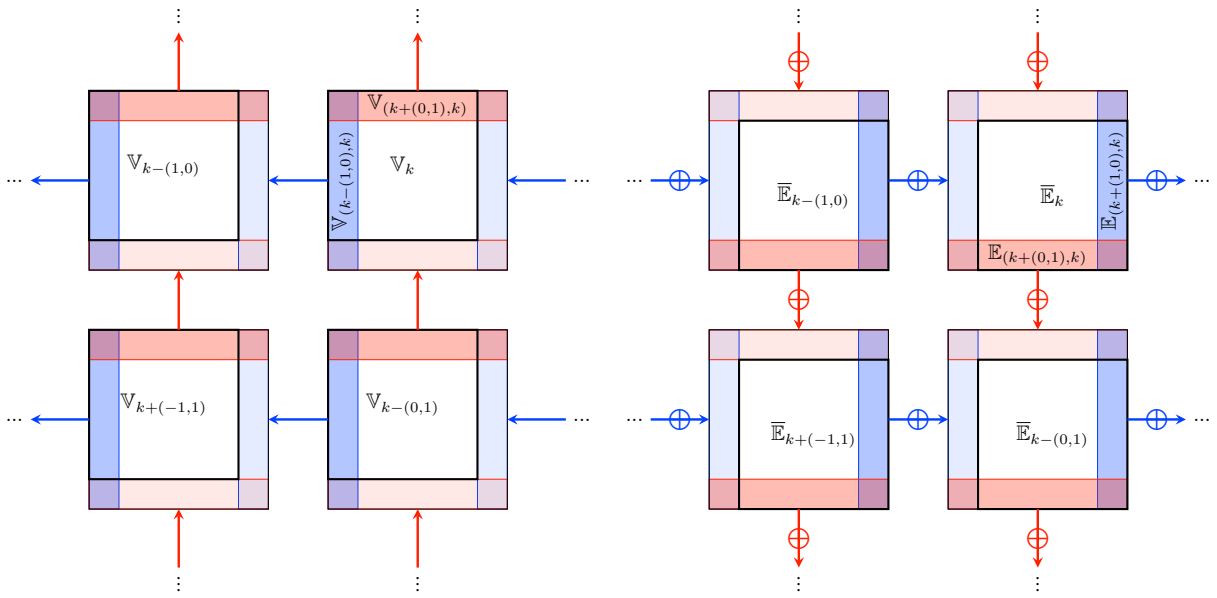
The proposed approach is applicable when a single operator is involved in the model (as in Section 5), but also when multiple operators are involved (see Appendix A), as in the present example. Note that a client-server architecture could also be considered. However, it would drastically increase the communication costs, as full-size variables would need to be duplicated on all workers or exchanged (see Section 5.3). Note that (6.2) indicates that the variables $(y_m)_{1\leqslant m\leqslant M}$ are assumed independent. This implies that the observations can be partitioned into K statistically independent blocks $\boldsymbol{y} = (\boldsymbol{y}_k)_{1\leqslant k\leqslant K}$, with $\boldsymbol{y}_k \in \mathbf{G}_k$ a block of observations to be stored on the worker k. In this case, each worker can be assigned a block of observations $\boldsymbol{y}_k$ and a corresponding block of parameters $\boldsymbol{x}_k$. Thus, no observation needs to be exchanged between the workers.

We propose to use the proposed distributed SGS Algorithm 5with $C = 2$ linear operators. In addition, for every $t \in \{0, \dots, T\}$ and $k \in \{1, \dots, K\}$, we use PSGLA transitions to compute $(\boldsymbol{z}_{1,k}^{(t+1)}, \boldsymbol{z}_{2,k}^{(t+1)})$, while $(\boldsymbol{u}_{1,k}^{(t+1)}, \boldsymbol{u}_{2,k}^{(t+1)})$ are drawn from their conditional distribution. More precisely, we have, for $c \in \{1, 2\}$,

$$\boldsymbol{z}_{c,k}^{(t+1)} = \mathrm{prox}_{\eta_c, g_{c,k}}\left(\boldsymbol{z}_{c,k}^{(t)} - \eta_c \alpha_c^{-2}\left(\boldsymbol{z}_{c,k}^{(t)} - \boldsymbol{v}_{c,k}^{(t+1)} + \boldsymbol{u}_{c,k}^{(t)}\right) + \sqrt{2\eta_c}\boldsymbol{\xi}_{c,k}^{(t)}\right), \tag{6.7}$$

where $g_{c,k} = \sum_{m\in\overline{\mathbb{E}}_{c,k}} g_{c,m}$, $\eta_c \in]0, \alpha_c^{-2}[$, and $\xi_{c,k}^{(t)} \sim \mathcal{N}(0, \mathbf{I})$. The proximity operators involved in the resulting algorithm can be found, e.g., in [17, 20]. The distributed implementation of $(\boldsymbol{D}_c)_{1\leqslant c\leqslant 2}$ and $(\boldsymbol{D}_c^*)_{1\leqslant c\leqslant 2}$ is detailed below, using a 2D Cartesian grid of K workers.

Distributed implementation of $\boldsymbol{D}_1$ Figure 6 illustrates the preliminary communications required by the distributed implementation of $\boldsymbol{D}_1$. Each worker $k \in \{1, \dots, K\}$ needs to collect $(\boldsymbol{x}_{1,(k,k')})_{k'\in\mathcal{R}_k}$, as illustrated in Figure 6(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 6(a), dark blue areas), and receives vertices from the neighbour on its right (Figure 6(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 6(a), dark and light red areas, of width L_1). The local opera-



(a) Distributed implementation of $\boldsymbol{D}_1$ (b) Distributed implementation of $\boldsymbol{D}_1^*$

Figure 6: Communication patterns involved in the distributed implementation of (a) $\boldsymbol{D}_1$ and (b) $\boldsymbol{D}_1^*$. 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 (5.13).

tor $(\boldsymbol{D}_{1,m,\overline{\mathcal{W}}_{1,m}})_{m\in\mathbb{E}_{1,\mathcal{R}_k}}$ 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 $(\boldsymbol{x}_{\overline{\mathcal{W}}_{1,m}})_{m\in\mathbb{E}_{\mathcal{R}_k}}$ are valid).

Distributed implementation of $\boldsymbol{D}_1^*$ The distributed implementation of $\boldsymbol{D}_1^*$ is similar to that of $\boldsymbol{D}_1$. 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 6(b), dark blue areas), and aggregates weights from the neighbour on its left (Figure 6(b), light blue areas). Similar communications occurs along the vertical axis with the bottom and top neighbours of the worker k (Figure 6(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 are used.

Distributed implementation of $\boldsymbol{D}_2$ The distributed implementation of $\boldsymbol{D}_2$ requires the same communication pattern as $\boldsymbol{D}_1$, successively exchanging messages with width 1 along the horizontal and vertical directions. The operator $(\boldsymbol{D}_{2,m,\overline{\mathcal{W}}_{2,m}})_{m\in\mathbb{E}_{2,\mathcal{R}_k}}$ corresponds to a local discrete gradient operator, using the boundaries retrieved during the communication step. Note that, for $k \in \{1,\dots,K\}$ and $k' \in \mathcal{R}_k$, the set of vertices $\mathbb{V}_{2,(k,k')}$ to be communicated by the worker k' to k is such that $\mathbb{V}_{2,(k,k')} \subset \mathbb{V}_{1,(k,k')}$. The total number of elements to be communicated in the algorithm is thus reduced, as vertices required by the distributed implementation of $\boldsymbol{D}_2$ already need to be communicated for $\boldsymbol{D}_1$.

Distributed implementation of $\boldsymbol{D}_2^*$ The distributed implementation of $\boldsymbol{D}_2^*$ requires the same communication pattern as $\boldsymbol{D}_1^*$, successively exchanging messages with width 1 along the horizontal and vertical directions. A local adjoint discrete 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 SGS algorithm from [35].

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]. Codes to reproduce the experiments will be available at https://gitlab.cristal.univ-lille.fr/pthouven/dsgs upon acceptance of the paper.

The proposed approach is compared with the SGS algorithm proposed in [35]. The latter relies on a different splitting strategy compared to Section 6.1, using $C = 3$ operators (see [35] 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 is assessed in terms of average runtime per iteration (with associated standard deviation) and quality of both the *minimum mean square error* (MMSE) and *maximum a posteriori* (MAP) estimators. The estimators are denoted $\boldsymbol{x}_{\text{MMSE}}$ and $\boldsymbol{x}_{\text{MAP}}$, respectively. Reconstruction quality is quantified with the structural similarity index (SSIM) [39] and the signal-to-noise ratio (SNR) expressed in dB. Associated 95% credibility intervals (CIs) are also reported.

The sampler from [35] has been applied with $\kappa = 1$ and $((\alpha_c^2, \beta_c^2)_{1 \leqslant c \leqslant 3}) = \mathbf{1}_6$, where $\mathbf{1}_Q \in \mathbb{R}^Q$ is a vector with entries all equal to 1. The proposed approach uses $\kappa = 1$ and $((\alpha_c^2, \beta_c^2)_{1 \leqslant c \leqslant 2}) = \mathbf{1}_4$. For both algorithms, $N_{\text{MC}} = 5 \times 10^3$ samples have been generated to form $\boldsymbol{x}_{\text{MMSE}}$, $\boldsymbol{x}_{\text{MAP}}$ and the 95% CIs, discarding $N_{\text{bi}} = 2 \times 10^3$ burn-in samples.

6.3.2 Experiment results

Estimation quality Using $K = 1$ worker, the proposed approach is compared with [35]. Ground truth images with different maximum intensity levels $\overline{x}_{\max} = \max_{1 \leqslant n \leqslant N} \overline{x}_n$ have been considered. The values $\overline{x}_{\max} \in \{20, 30\}$ have been adopted for the following datasets:

(i) `house` image ($N = 256^2$) with a normalized Gaussian kernel of size $L \in \{3^2, 7^2\}$;

(ii) `peppers` image ($N = 512^2$) with 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 [35]. In addition, the computing time required by the proposed sampler is between 1.5 and 2 times smaller than [35]. This discrepancy comes from the difference in the splitting strategies considered by the two methods. In particular, the splitting approach proposed in [35] 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 seen in the difference in quality of the estimators. The MMSE estimator reported in Figure 7 for [35] appears much smoother compared to the proposed algorithm for the same regularization parameter

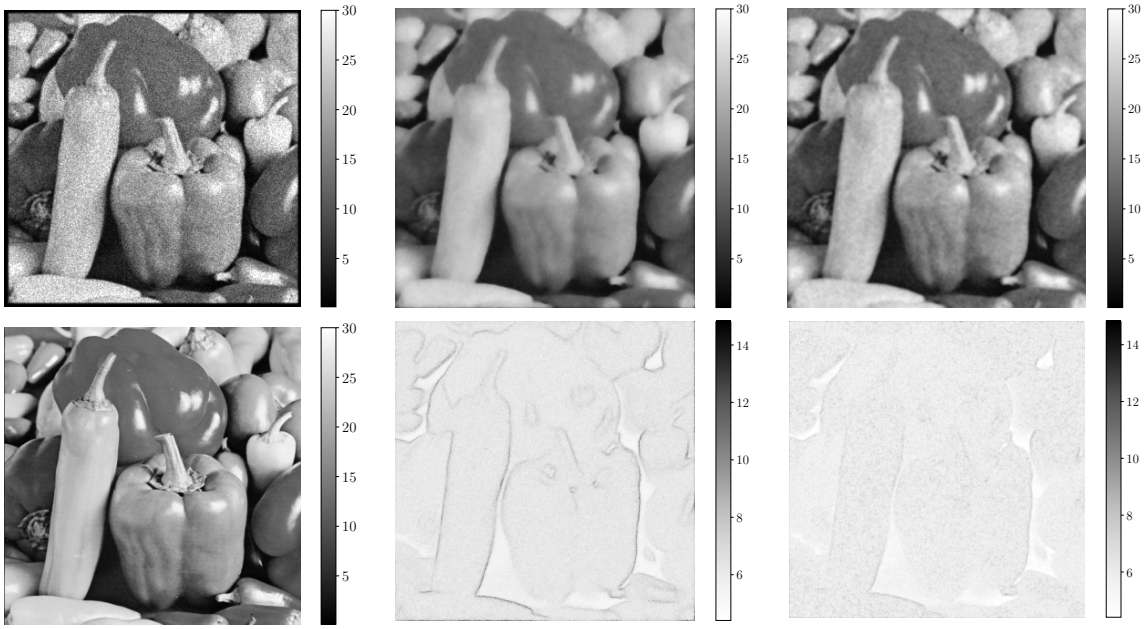
κ. Note that the uncertainty level of the proposed approach is slightly lower and appears more diffuse than [35].

	Dataset	Algo.	SNR($\boldsymbol{x}_{\text{MMSE}}$)	SNR($\boldsymbol{x}_{\text{MAP}}$)	SSIM($\boldsymbol{x}_{\text{MMSE}}$)	SSIM($\boldsymbol{x}_{\text{MAP}}$)	Time per iter. ($\times 10^{-2}$ s)	Runtime ($\times 10^{2}$ s)
$\overline{x}_{\max} = 20$	House ($L = 3^2$)	[35]	18.37	15.08	0.60	0.20	5.59 (0.12)	1.68
		Proposed	20.21	16.18	0.60	0.26	**2.49 (0.09)**	**0.75**
	House ($L = 7^2$)	[35]	18.00	14.92	0.60	0.20	8.52 (0.22)	2.56
		Proposed	19.86	15.98	0.59	0.24	**4.88 (0.13)**	**1.47**
	Peppers ($L = 7^2$)	[35]	18.98	15.08	0.66	0.25	16.71 (0.38)	5.01
		Proposed	20.52	16.07	0.67	0.30	**7.73 (0.33)**	**2.32**
	Peppers ($L = 15^2$)	[35]	18.90	15.07	0.66	0.25	36.68 (0.33)	11.01
		Proposed	20.52	16.04	0.66	0.30	**23.32 (0.40)**	**7.00**
$\overline{x}_{\max} = 30$	House ($L = 3^2$)	[35]	18.23	16.46	0.64	0.29	4.71 (0.10)	1.41
		Proposed	20.18	17.84	0.66	0.34	**2.03 (0.05)**	**0.61**
	House ($L = 7^2$)	[35]	17.89	16.25	0.64	0.28	7.01 (0.10)	2.10
		Proposed	19.80	17.55	0.65	0.33	**3.93 (0.10)**	**1.18**
	Peppers ($L = 7^2$)	[35]	19.03	16.77	0.69	0.35	17.04 (0.45)	5.11
		Proposed	20.71	17.96	0.71	0.40	**10.26 (1.22)**	**3.08**
	Peppers ($L = 15^2$)	[35]	19.00	16.72	0.69	0.35	36.81 (0.44)	11.04
		Proposed	20.74	17.98	0.71	0.41	**23.22 (0.34)**	**6.97**

Table 3: Comparison between [35] and the proposed approach with $K = 1$. Datasets have been generated from ground truth images with different maximum intensity $\overline{x}_{\max} \in \{20, 30\}$ and convolution kernel sizes $L \in \{3, 7, 15\}$. Results are reported in terms of estimation quality, average runtime per iteration (with standard deviation) and total runtime.

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 $\overline{x}_{\max} = 30$ for the `peppers` dataset ($N = 512^2$).

Tables 4 and 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 [35] for a client-server architecture with $K = 4$ cores is 1.48×10^{-1} s when $L = 7^2$ (1.04×10^{-1} s for TV-related terms), and 3.41×10^{-1} s when $L = 15^2$ (1.02×10^{-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.



(a) Observations and ground truth

(b) MMSE estimator and 95% CI [35]

(c) MMSE estimator and 95% CI (prop., $K = 1$)

Figure 7: Estimators and 95% credibility intervals for `peppers`, with $\overline{x}_{\max} = 30$ and $L = 15^2$.

Weak scaling 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 $\overline{x}_{\max} = 30$, using $(N, L) \in \{(256^2, 3^2), (512^2, 7^2), (1022^2, 11^2)\}$. 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 SGS has been introduced to efficiently solve large scale imaging inverse problems. The approach leverages the approximate data augmentation scheme AXDA [37, 28] to efficiently handle composite functions involving linear operators. A

K	SNR($\boldsymbol{x}_{\text{MMSE}}$)	SNR($\boldsymbol{x}_{\text{MAP}}$)	SSIM($\boldsymbol{x}_{\text{MMSE}}$)	SSIM($\boldsymbol{x}_{\text{MAP}}$)	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 $\overline{x}_{\max} = 30$ and kernel size $L = 7^2$. Performance is reported in terms of estimation quality, time per iteration and speedup.

K	SNR($\boldsymbol{x}_{\text{MMSE}}$)	SNR($\boldsymbol{x}_{\text{MAP}}$)	SSIM($\boldsymbol{x}_{\text{MMSE}}$)	SSIM($\boldsymbol{x}_{\text{MAP}}$)	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 $\overline{x}_{\max} = 30$ and kernel size $L = 15^2$. Performance is reported in terms of estimation quality, time per iteration and speedup.

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 split Gibbs sampler.

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 from [35]. 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

(N, L, K)	SNR($\boldsymbol{x}_{\text{MMSE}}$)	SNR($\boldsymbol{x}_{\text{MAP}}$)	SSIM($\boldsymbol{x}_{\text{MMSE}}$)	SSIM($\boldsymbol{x}_{\text{MAP}}$)	Time per iter. ($\times 10^{-2}$ s)	Scaled speedup	Runtime ($\times 10^2$ s)
$(256^2, 3^2, \mathbf{1})$	20.18	17.84	0.66	0.34	2.03 (0.05)	**1.00**	**0.61**
$(512^2, 7^2, \mathbf{4})$	23.86	19.56	0.74	0.34	2.13 (0.08)	**3.81**	**0.64**
$(1022^2, 11^2, \mathbf{16})$	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 $\overline{x}_{\max} = 30$. The reconstruction quality is reported with the time per iteration, the scaled speedup and the runtime.

credibility intervals.

Note that the proposed distributed block-coordinate SGS 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 SGS for multiple composite terms

A.1 AXDA for multiple composite terms

Following [37], AXDA can be generalized to address distributions of the form (1.3), which involve multiple composite terms. We will use similar ideas to generalize the proposed approach, leveraging the approximation (1.4) associated with distribution (1.3). In [37], the authors show that Proposition 2.1 holds in this context, under the same conditions on ϕ_{c,α_c} and ψ_{c,β_c} as (2.4)-(2.5).

Using an approach similar to the one described in Section 2.2, we can design a PSGLA within Gibbs sampler to approximately draw samples from (1.4), generalizing Algorithm 2. The resulting algorithm is reported in Algorithm 4, where $(\boldsymbol{w}^{(t)})_{0\leqslant t\leqslant T}$ is a sequence of independent and identically distributed (i.i.d) standard Gaussian random variables in $\mathbf{H}$, $\gamma \in]0, \lambda^{-1}[$, $\lambda = \lambda_h + \| \sum_{c=1}^{C} \boldsymbol{D}_c/\alpha_c \|^2$, and, for every $c \in \{1, \ldots, C\}$,

$$(\forall c \in \{1,\ldots,C\}) \quad \pi_{c,\alpha_c}(\boldsymbol{z}_c \mid \boldsymbol{v}_c, \boldsymbol{u}_c) \propto \exp\left(-g_c(\boldsymbol{z}_c) - \phi_{c,\alpha_c}(\boldsymbol{v}_c, \boldsymbol{z}_c - \boldsymbol{u}_c)\right), \tag{A.1}$$

with $\boldsymbol{v}_c = \boldsymbol{D}_c \boldsymbol{x}$.

Algorithm 4: Proposed multi-term SGS (with $C \geqslant 1$ composite terms).

Input: $\boldsymbol{x}^{(0)} \in \mathbf{H}$, $(\boldsymbol{z}_c^{(0)}, \boldsymbol{u}_c^{(0)}) \in \mathbf{G}_c^2$ and $(\alpha_c, \beta_c) \in]0, +\infty[^2$ for $c \in \{1, \ldots, C\}$, $\gamma \in \left]0, \left(\lambda_h + \| \sum_{c=1}^{C} \boldsymbol{D}_c/\alpha_c \|^2\right)^{-1}\right[$

for $t = 0$ **to** T **do**

$$\boldsymbol{x}^{(t+1)} = \operatorname{prox}_{\gamma f}\left(\boldsymbol{x}^{(t)} - \gamma \nabla h(\boldsymbol{x}^{(t)}) - \gamma \sum_{c=1}^{C} \boldsymbol{D}_c^* \nabla \phi_{c,\alpha_c}(\cdot, \boldsymbol{z}_c^{(t)} - \boldsymbol{u}_c^{(t)})(\boldsymbol{v}_c^{(t)}) + \sqrt{2\gamma}\boldsymbol{w}^{(t)}\right);$$

  for $i = 1$ **to** C **do**

$$\boldsymbol{v}_c^{(t+1)} = \boldsymbol{D}_c \boldsymbol{x}^{(t+1)},$$

$$\boldsymbol{z}_c^{(t+1)} \sim \pi_{c,\alpha_c}\left(\boldsymbol{z}_c \mid \boldsymbol{v}_c^{(t+1)}, \boldsymbol{u}_c^{(t)}\right);$$

$$\boldsymbol{u}_c^{(t+1)} \sim \mathcal{N}\Big(\frac{\beta_c^2}{\alpha_c^2 + \beta_c^2}(\boldsymbol{z}_c^{(t+1)} - \boldsymbol{v}_c^{(t+1)}), \frac{\alpha_c^2 + \beta_c^2}{\alpha_c^2 \beta_c^2}\mathbf{I}\Big);$$

Output: $(\boldsymbol{x}^{(t)})_{1\leqslant t\leqslant T}$, $(\boldsymbol{z}_c^{(t)}, \boldsymbol{u}_c^{(t)})_{1\leqslant t\leqslant T, 1\leqslant i\leqslant C}$

Notation	Definition	Given by operator $\boldsymbol{D}_c$	Given by user
$\boldsymbol{e}_c = (e_{c,m})_{1 \leqslant m \leqslant M_c}$	Hyperedges of $\mathcal{H}_c$	✓	
$e_{c,m} \subset \{1, \dots, N\}$	Vertex indices in hyperedge m of $\mathcal{H}_c$	✓	
$k_{c,m} \in \{1, \dots, K\}$	Worker associated with m-th hyperedge $e_{c,m}$ (chosen by the user). $k_{c,m}$ must satisfy $e_{c,m} \cap \mathbb{V}_{k_{c,m}} \neq \varnothing$		✓
$\mathcal{W}_{c,m} \subset \{1, \dots, K\} \setminus \{k_{c,m}\}$	Set of all workers but $k_{c,m}$, containing vertices from $e_{c,m}$	✓	
$\overline{\mathcal{W}}_{c,m} \subset \{1, \dots, K\}$	$\overline{\mathcal{W}}_{c,m} = k_{c,m} \cup \mathcal{W}_{c,m}$	✓	
$\mathbb{V}_{c,(k,k')} \subset \mathbb{V}_{k'}$	Indices of vertices sent from worker k' to worker k	✓	✓
$\mathbb{V}_{\mathcal{W}_{c,m}} \subset \{1, \dots, N\} \setminus \mathbb{V}_{k_{c,m}}$	$\mathbb{V}_{\mathcal{W}_{c,m}} = \bigcup_{k' \in \mathcal{W}_{c,m}} \mathbb{V}_{(k_{c,m},k')}$ the set of vertex indices that will be communicated to worker $k_{c,m}$ from all workers $k' \in \mathbb{V}_{c,m}$	✓	✓
$\mathbb{V}_{\overline{\mathcal{W}}_{c,m}} \subset \{1, \dots, N\}$	$\mathbb{V}_{\overline{\mathcal{W}}_{c,m}} = \mathbb{V}_{k_{c,m}} \cup \mathbb{V}_{\mathcal{W}_{c,m}}$ the set of vertex indices necessary to perform computations associated with $k_{c,m}$	✓	✓
$\mathbb{E}_{c,k} \subset \{1, \dots, M_c\}$	Indices of hyperedges only containing vertices stored on worker k	✓	✓
$\mathbb{E}_{c,(k,k')} \subset \{1, \dots, M_c\}$	Indices of hyperedges containing vertices sent from worker k' to worker k	✓	✓
$\mathbb{E}_{c,\mathcal{R}_k} \subset \{1, \dots, M_c\}$	$\mathbb{E}_{c,\mathcal{R}_k} = \cup_{k' \in \mathcal{R}_k} \mathbb{E}_{c,(k,k')}$ set of all hyperedges containing vertices communicated to worker k	✓	✓
$\overline{\mathbb{E}}_{c,k} \subset \{1, \dots, M_c\}$	$\overline{\mathbb{E}}_{c,k} = \mathbb{E}_{c,k} \cup \mathbb{E}_{c,\mathcal{R}_k}$, such that $(\overline{\mathbb{E}}_{c,k})_{1 \leqslant k \leqslant K}$ is a partition of $\{1, \dots, M_c\}$	✓	✓

Table 7: Summary of the set notation used to define the hypergraph structure associated with the operators $\boldsymbol{D}_c$, for $c \in \{1, \dots, C\}$. This notation generalizes the one from Table 1.

A.2 Distributed multi-term SGS

This section introduces a distributed version of Algorithm 4, using the same approach as in Sections 4 and 5. We consider the distribution (1.3), with its AXDA approximation (1.4).

For every $c \in \{1, \dots, C\}$, let $\mathbf{G}_c = \mathbb{R}^{\overline{M}_c}$ such that $\mathbf{G}_c = \mathsf{G}_{c,1} \times \ldots \times \mathsf{G}_{c,M_c}$, where for every $m \in \{1, \dots, M_c\}$, $\mathsf{G}_{c,m} = \mathbb{R}^{M_{c,m}}$, with $\overline{M}_c = \sum_{m=1}^{M_c} M_{c,m}$. An element of $\mathbf{G}_c$ is denoted by $\boldsymbol{u}_c = (u_{c,m})_{1 \leqslant m \leqslant M_c}$, where, for every $m \in \{1, \dots, M_c\}$, $u_{c,m} \in \mathsf{G}_{c,m}$.

For every $c \in \{1, \dots, C\}$, the linear operator $\boldsymbol{D}_c = (D_{c,m,n})_{1 \leqslant m \leqslant M_c, 1 \leqslant n \leqslant N}$ defines a hypergraph structure $\mathcal{H}_c$ as described in Section 4. We thus consider C 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 Tables 7 and 8 generalize those introduced in Section 4. For every $k \in \{1, \dots, K\}$, let $\mathbf{G}_{c,k} = \times_{m \in \overline{\mathbb{E}}_{c,k}} \mathsf{G}_{c,m}$ such that $\mathbf{G}_c = \times_{1 \leqslant k \leqslant K} \mathbf{G}_{c,k}$.

We assume that, for every $c \in \{1, \dots, C\}$, the functions g_c, ϕ_{c,α_c} and ψ_{c,β_c} satisfy the same assumptions as the functions g, ϕ_α, ψ_β given in Theorems 2.2 and 4.4. Then, for every $c \in$

Notation	Definition
$\boldsymbol{D}_c \colon \mathbf{H} \to \mathbf{G}_c$	$\boldsymbol{D}_c = (D_{c,m,n})_{1 \leqslant m \leqslant M_c, 1 \leqslant n \leqslant N}$ linear operator defining the ith hypergraph
$\boldsymbol{u}_c, \boldsymbol{v}_c \in \mathbf{G}_c$	Hyperedge weights
$\boldsymbol{v}_c = (v_{c,m})_{1 \leqslant m \leqslant M_c} \in \mathbf{G}_c$	
$\boldsymbol{v}_{c,k} \in \mathbf{G}_{c,k}$	$\boldsymbol{v}_{c,k} = (v_{c,m})_{m \in \overline{\mathbb{E}}_{c,k}}$ hyperedge weights stored on worker k
$\boldsymbol{D}_{c,m,k} \colon \mathbf{H}_{c,k} \to \mathsf{G}_{c,m}$	$\boldsymbol{D}_{c,m,k} = (D_{c,m,n})_{n \in \mathbb{V}_{c,k}}$, for $m \in \mathbb{E}_{c,k}$, subpart of $\boldsymbol{D}_c$ stored on worker k, associated with hyperedges containing vertices only on worker k
$\boldsymbol{x}_{c,(k,k')} \in \bigtimes_{n \in \mathbb{V}_{c,(k,k')}} \mathsf{H}_n$	$\boldsymbol{x}_{c,(k,k')} = (x_n)_{n \in \mathbb{V}_{c,(k,k')}}$ vertex values sent from worker k' to k
$\boldsymbol{D}_{c,m,(k,k')} \colon \bigtimes_{n \in \mathbb{V}_{c,(k,k')}} \mathsf{H}_n \to \mathsf{G}_{c,m}$	$\boldsymbol{D}_{c,m,(k,k')} = (D_{c,m,n})_{n \in \mathbb{V}_{c,(k,k')}}$, for $m \in \mathbb{E}_{c,(k,k')}$, part of $\boldsymbol{D}_c$ stored on worker k, related to hyperedges containing vertices overlapping k and k'
$\boldsymbol{x}_{\overline{\mathcal{W}}_{c,m}} \in \bigtimes_{n \in \mathbb{V}_{\overline{\mathcal{W}}_{c,m}}} \mathsf{H}_n$	$\boldsymbol{x}_{\overline{\mathcal{W}}_{c,m}} = (x_n)_{n \in \mathbb{V}_{\overline{\mathcal{W}}_{c,m}}}$, concatenation of vertex values stored on worker $k_{c,m}$, and those sent from all workers $k' \in \mathcal{W}_{c,m}$ to $k_{c,m}$
$\boldsymbol{D}_{c,m,\overline{\mathcal{W}}_{c,m}} \colon \bigtimes_{n \in \mathbb{V}_{\overline{\mathcal{W}}_{c,m}}} \mathsf{H}_n \to \mathsf{G}_{c,m}$	$\boldsymbol{D}_{c,m,\overline{\mathcal{W}}_{c,m}} = (D_{c,m,n})_{n \in \mathbb{V}_{\overline{\mathcal{W}}_{c,m}}}$, for $m \in \mathbb{E}_{c,\mathcal{R}_k}$, subpart of $(D_{c,m,n})_{1 \leqslant n \leqslant N}$ corresponding to vertices stored on worker $k_{c,m}$ and vertices overlapping worker $k_{c,m}$ and other workers of $\mathcal{W}_{c,m}$

Table 8: Notation used for the variables associated with the hypergraph induced by the operators $\boldsymbol{D}_c$, for $c \in \{1, \dots, C\}$. This notation generalizes the one provided in Table 2.

$\{1, \dots, C\}$, there exists a permutation $\varrho_c \colon \mathbf{G}_c \to \mathbf{G}_c$ such that, for every $\boldsymbol{x} \in \mathbf{H}$,

$$\boldsymbol{D}_c \boldsymbol{x} = \Big((D_{c,m,n})_{1 \leqslant n \leqslant N} \boldsymbol{x} \Big)_{1 \leqslant m \leqslant M_c} = \varrho_c \left(\begin{pmatrix} (\boldsymbol{D}_{c,m,k})_{m \in \mathbb{E}_{c,k}} \boldsymbol{x}_k \\ \left(\boldsymbol{D}_{c,\overline{\mathcal{W}}_{c,m}} \boldsymbol{x}_{\overline{\mathcal{W}}_{c,m}} \right)_{m \in \mathbb{E}_{c,\mathcal{R}_k}} \end{pmatrix}_{1 \leqslant k \leqslant K} \right), \tag{A.2}$$

and, for every $\boldsymbol{u}_c \in \mathbf{G}_c$, we have

$$g_c(\boldsymbol{u}_c) = \sum_{k=1}^{K} \Big(\sum_{m \in \overline{\mathbb{E}}_{c,k}} g_{c,m}(u_{c,m}) \Big),$$

$$\phi_{c,\alpha_c}(\boldsymbol{D}_c \boldsymbol{x}, \boldsymbol{u}_c) = \sum_{k=1}^{K} \Big(\sum_{m \in \mathbb{E}_{c,k}} \phi_{c,m,\alpha_c} \left(\boldsymbol{D}_{c,m,k} \boldsymbol{x}_{c,k}, u_{c,m} \right) + \sum_{m \in \mathbb{E}_{c,\mathcal{R}_k}} \phi_{c,m,\alpha_c} \left(\boldsymbol{D}_{\mathbb{V}_{c,m}} \boldsymbol{x}_{\mathbb{V}_{c,m}}, u_{c,m} \right) \Big),$$

$$\psi_{c,\beta_c}(\boldsymbol{u}) = \sum_{k=1}^{K} \Big(\sum_{m \in \overline{\mathbb{E}}_{c,k}} \psi_{c,m,\beta_c} \left(u_{c,m} \right) \Big).$$

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

Proposition A.1 *Assume that, for every $c \in \{1, \dots, C\}$, each operator $\boldsymbol{D}_c$ is split over workers $\{1, \dots, K\}$ such that, for every $k \in \{1, \dots, K\}$, $(\boldsymbol{D}_{c,m,k})_{m \in \overline{\mathbb{E}}_{c,k}}$ is stored on worker k. Let, for every $k \in \{1, \dots, K\}$, $\boldsymbol{x}_k^{(0)} \in \mathbf{H}_k$, $\boldsymbol{z}_{c,k}^{(0)} \in \mathbf{G}_{c,k}$, and $\boldsymbol{u}_{c,k}^{(0)} \in \mathbf{G}_{c,k}$. Let $(\boldsymbol{x}^{(t)})_{1 \leqslant t \leqslant T}$ and $(\boldsymbol{z}_c^{(t)}, \boldsymbol{u}_c^{(t)})_{1 \leqslant c \leqslant C, 1 \leqslant t \leqslant T}$ be samples generated by Algorithm 5, where $\gamma \in \left] 0, \left(\lambda_h + \sum_{c=1}^{C} (\|\boldsymbol{D}_c\|^2 / \alpha_c^2) \right)^{-1} \right[$, for every $k \in \{1, \dots, K\}$, $(\boldsymbol{w}_k^{(t)})_{1 \leqslant t \leqslant T}$ is a sequence of i.i.d. standard Gaussian random variables in $\mathbf{H}_k$. In addi-*

tion, for every $c \in \{1, \dots, C\}$*,*

$$\pi_{c,k,\alpha_c}\left(\boldsymbol{z}_{c,k} \mid \boldsymbol{v}_{c,k}, \boldsymbol{u}_{c,k}\right) \propto \exp\Bigg(- \sum_{m \in \mathbb{E}_{c,k}} \left(g_{c,m}(z_{c,m}) + \phi_{c,m,\alpha_c}(\boldsymbol{D}_{c,m,k}\boldsymbol{x}_k, z_{c,m} - u_{c,m})\right) - \sum_{m \in \mathbb{E}_{c,\mathcal{R}_k}} \left(g_{c,m}(z_{c,m}) + \phi_{c,m,\alpha_c}(\boldsymbol{D}_{c,m,\overline{\mathcal{W}}_{c,m}}\boldsymbol{x}_{\overline{\mathcal{W}}_{c,m}}, z_{c,m} - u_{c,m})\right)\Bigg),$$

where, for every $\boldsymbol{x} \in \mathsf{H}$*,* $\boldsymbol{v}_{c,k} = (v_{c,m})_{m \in \overline{\mathbb{E}}_{c,k}} = \varrho_c \begin{pmatrix} (\boldsymbol{D}_{c,m,k}\boldsymbol{x}_k)_{m \in \mathbb{E}_{c,k}} \\ (\boldsymbol{D}_{c,m,\overline{\mathcal{W}}_{c,m}}\boldsymbol{x}_{\overline{\mathcal{W}}_{c,m}})_{m \in \mathbb{E}_{c,\mathcal{R}_k}} \end{pmatrix}$.
Then Algorithm 5 is equivalent to Algorithm 4.

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 Sci., 2 (2009), pp. 183–202.

[3] J. Birdi, A. Repetti, and Y. Wiaux., *Sparse interferometric Stokes imaging under polarization constraint (Polarized SARA)*, 478 (2018), pp. 4442–4463.

[4] J. Bolte, S. Sabach, and M. Teboulle, *Proximal alternating linearized minimization for nonconvex and nonsmooth problems*, Math. Program., 1-2 (2014), pp. 459–494.

[5] X. Cai, M. Pereyra, and J. D. McEwen, *Uncertainty quantification for radio interferometric imaging – I. Proximal MCMC methods*, 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*, 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.

Algorithm 5: Proposed distributed SGS with multiple composite terms $C > 1$.

for $k = 1$ **to** K **do**
  for $k' \in \mathcal{S}_k$ **do**
    Send $(x_n^{(0)})_{n \in \cup_{c=1}^{C} \mathbb{V}_{c,(k',k)}}$ to worker k';
  for $k' \in \mathcal{R}_k$ **do**
    Receive $(x_n^{(0)})_{n \in \cup_{c=1}^{C} \mathbb{V}_{c,(k,k')}}$ from worker k';
  for $c = 1$ **to** C **do**
    $\boldsymbol{v}_{c,k}^{(0)} = \varrho_c \begin{pmatrix} (\boldsymbol{D}_{c,m,k} \boldsymbol{x}_k^{(0)})_{m \in \mathbb{E}_{c,k}} \\ (\boldsymbol{D}_{c,m,\overline{\mathcal{W}}_{c,m}} \boldsymbol{x}_{\overline{\mathcal{W}}_{c,m}}^{(0)})_{m \in \mathbb{E}_{c,\mathcal{R}_k}} \end{pmatrix}$;
for $t = 0$ **to** T **do**
  for $k = 1$ **to** K **do**
    for $c = 1$ **to** C **do**
      $(d_{c,m}^{(t)})_{m \in \overline{\mathbb{E}}_{c,k}} = \left(\phi'_{c,m,\alpha_i}(\cdot, z_{c,m}^{(t)} - u_{c,m}^{(t)})(v_{c,m}^{(t)})\right)_{m \in \overline{\mathbb{E}}_{c,k}}$;
    for $k' \in \mathcal{R}_k$ **do**
      $\widetilde{\boldsymbol{d}}_{(k',k)}^{(t)} = \sum_{c=1}^{C} \sum_{m \in \mathbb{E}_{c,(k',k)}} \boldsymbol{D}_{c,m,k'}^{*} d_{c,m}^{(t)}$;
      Send $\widetilde{\boldsymbol{d}}_{(k',k)}^{(t)}$ to worker k';
    for $k' \in \mathcal{S}_k$ **do**
      Receive $\widetilde{\boldsymbol{d}}_{(k,k')}^{(t)}$ from worker k';
    $\boldsymbol{\delta}_k^{(t)} = \sum_{c=1}^{C} \sum_{m \in \overline{\mathbb{E}}_{c,k}} \boldsymbol{D}_{c,m,k}^{*} d_{c,m}^{(t)} + \sum_{k' \in \mathcal{S}_k} \widetilde{\boldsymbol{d}}_{(k,k')}^{(t)}$;
    $\boldsymbol{x}_k^{(t+1)} = \operatorname{prox}_{\gamma f_k}\left(\boldsymbol{x}_k^{(t)} - \gamma \nabla h_k(\boldsymbol{x}_k^{(t)}) - \gamma \boldsymbol{\delta}_k^{(t)} + \sqrt{2\gamma}\boldsymbol{w}_k^{(t)}\right)$;
    for $k' \in \mathcal{S}_k$ **do**
      Send $(x_n^{(t+1)})_{n \in \cup_{c=1}^{C} \mathbb{V}_{c,(k',k)}}$ to worker k';
    for $k' \in \mathcal{R}_k$ **do**
      Receive $(x_n^{(t+1)})_{n \in \cup_{c=1}^{C} \mathbb{V}_{c,(k,k')}}$ from worker k';
    for $c = 1$ **to** C **do**
      $\boldsymbol{v}_{c,k}^{(t+1)} = \varrho_c \begin{pmatrix} (\boldsymbol{D}_{c,m,k} \boldsymbol{x}_k^{(t+1)})_{m \in \mathbb{E}_{c,k}} \\ (\boldsymbol{D}_{c,m,\overline{\mathcal{W}}_{c,m}} \boldsymbol{x}_{\overline{\mathcal{W}}_{c,m}}^{(t+1)})_{m \in \mathbb{E}_{c,\mathcal{R}_k}} \end{pmatrix}$;
      $\boldsymbol{z}_{c,k}^{(t+1)} \sim \pi_{c,\alpha_c,k}\left(\boldsymbol{z}_{c,k} \mid \boldsymbol{v}_{c,k}^{(t+1)}, \boldsymbol{u}_{c,k}^{(t)}\right)$;
      $\boldsymbol{u}_{c,k}^{(t+1)} \sim \mathcal{N}\left(\frac{\beta_i^2}{\alpha_i^2+\beta_i^2}(\boldsymbol{z}_{c,k}^{(t+1)} - \boldsymbol{v}_{c,k}^{(t+1)}), \frac{\alpha_i^2+\beta_i^2}{\alpha_i^2\beta_i^2}\mathbf{I}\right)$;

[10] P. L. COMBETTES AND J.-C. PESQUET, *Proximal splitting methods in signal processing*, in Fixed-Point Algorithms for Inverse Problems in Science and Engineering, 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 J. Optim., 25 (2015), pp. 1221–1248.

[12] P. L. COMBETTES AND B. C. VŨ, *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, proximable 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] J.-B. HIRIART-URRUTY AND C. LEMARÉCHAL, *Convex Analysis and Minimization Algorithms II*, Springer-Verlag Berlin Heidelberg, 1993, https://doi.org/10.1007/978-3-662-06409-2.

[19] G. L. JONES, G. O. ROBERTS, AND J. S. ROSENTHAL, *Convergence of conditional Metropolis-Hastings samplers*, 46 (2014), pp. 422–445.

[20] 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.

[21] L. LIMA PILLA, *Optimal Task Assignment for Heterogeneous Federated Learning Devices*, May 2021, pp. 661–670, https://doi.org/10.1109/IPDPS49936.2021.00074.

[22] 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.

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

[24] Y. E. NESTEROV, *A method for solving the convex programming problem with convergence rate* $O(1/k^2)$, Dokl. Akad. Nauk SSSR, 269 (1983), pp. 543–547.

[25] M. Pereyra, *Proximal Markov chain Monte Carlo algorithms*, Statist. Comput., 26 (2016), pp. 745–760.

[26] M. Pereyra, P. Schniter, É. Chouzenoux, J.-C. Pesquet, J.-Y. Tourneret, A. O. Hero, and S. McLaughlin, *A survey of stochastic simulation and optimization methods in signal processing*, IEEE J. Selected Topics Signal Process., 10 (2016), pp. 224–241, https://doi.org/10.1109/JSTSP.2015.2496908.

[27] 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.

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

[29] 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.

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

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

[32] 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*, 521 (2022), pp. 1–19.

[33] P.-A. Thouvenin, A. Repetti, and P. Chainais, *A versatile distributed MCMC algorithm for large scale inverse problems*, in Proc. Eur. Sig. and Image Proc. Conference, Belgrade, Serbia, 2022, pp. 2016–2020.

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

[35] M. Vono, N. Dobigeon, and P. Chainais, *Bayesian image restoration under poisson noise and log-concave prior*, in Proc. IEEE Int. Conf. Acoust., Speech Signal Process., Brighton, United Kingdom, 2019, pp. 1712–1716.

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

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

[38] B. C. Vũ, *A splitting algorithm for dual monotone inclusions involving cocoercive operators*, Adv. Comput. Math., 38 (2013), pp. 667–681.

[39] 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.