Distributed Algorithms for Scalable Proximity Operator Computation and Application to Video Denoising

Feriel Abboud, Marin Stamm, Emilie Chouzenoux, Jean-Christophe Pesquet, Hugues Talbot

Distributed Algorithms for Scalable Proximity Operator Computation and Application to Video Denoising

Feriel Abboud[a], Marin Stamm[b], Emilie Chouzenoux[c,*], Jean-Christophe Pesquet[c], Hugues Talbot[c]

[a] *WITBE, Les Collines de l'Arche, Immeuble Opéra. 92800 Puteaux, France*
[b] *ENSTA Paris, Institut Polytechnique de Paris, Palaiseau, France*
[c] *Center for Visual Computing, CentraleSupélec, Inria, University Paris-Saclay, Gif-sur-Yvette, France*

Abstract

Optimization problems arising in signal and image processing involve an increasingly large number of variables. In addition to the curse of dimensionality, another difficulty to overcome is that the cost function usually reads as the sum of several loss/regularization terms, which are non-necessarily smooth and possibly composed with large-size linear operators. Proximal splitting approaches are fundamental tools to address such problems, with demonstrated efficiency in many applicative fields. In this paper, we present a new distributed algorithm for computing the proximity operator of a sum of non-necessarily smooth convex functions composed with arbitrary linear operators. Our algorithm relies on a primal-dual splitting strategy, and benefits from established convergence guaranties. Each involved function is associated with a node of a hypergraph, with the ability to communicate with neighboring nodes sharing the same hyperedge. Thanks to this structure, our method can be efficiently implemented on modern parallel computing architectures, distributing the computations on multiple nodes or machines, with controlled requirements for synchronization steps. Good numerical performance and scalability properties are

*Corresponding author
Email addresses: feriel.abboud@gmail.com (Feriel Abboud), marin.stamm@ensta-paris.fr (Marin Stamm), emilie.chouzenoux@centralesupelec.fr (Emilie Chouzenoux), jean-christophe.pesquet@centralesupelec.fr (Jean-Christophe Pesquet), hugues.talbot@centralesupelec.fr (Hugues Talbot)

demonstrated on a problem of video sequence denoising. Our code implemented in Julia is made available at `https://github.com/MarinENSTA/distributed_julia_denoising`.

Keywords: Convex optimization, proximal methods, distributed algorithm, parallel programming, video restoration.

1. Introduction

Numerous problems in data science such as video restoration require the processing of large datasets. Optimal processing are often obtained by solving nonsmooth optimization problems, for which proximity operators appear as fundamental tools. In this context, it is necessary to propose parallel and distributed methods to compute proximity operators involved in the solution of high-dimensional problems, especially when the objective function is the sum of several convex non-necessarily smooth functions [1, 2]. In the general case, a closed form expression of the proximity operator of such composite term does not exist, and developing iterative strategies becomes necessary.

Primal-dual splitting methods are prominently used when dealing with convex optimization problems where large-size linear operators are involved [3, 4, 5, 6]. The main advantage of many of these algorithms is that they do not require computing the inverse of these linear operators, which makes this class of algorithms well suited for large-scale problems encountered in various application fields [7, 8, 9]. Primal-dual techniques are based on several well-known strategies such as the Forward-Backward iteration [10, 11], the Forward-Backward-Forward iteration [12, 13], the Douglas-Rachford algorithm [14, 15], or the Alternating Direction Method of Multipliers [16, 17, 18, 19, 20]. Moreover, primal-dual algorithms can be combined with a block-coordinate approach, where at each iteration only a few blocks are activated following a specific selection rule [21, 22]. These algorithms can achieve fast convergence speed with contained memory requirement. Both stochastic [23, 24] and deterministic [25, 26] versions of these have been used in image processing and machine learning applications.

In the latter context, algorithms based on a dual Forward-Backward approach are often referred to as dual ascent methods.

The aforementioned algorithms were originally made available with single-node implementations, which may be suboptimal or even unsuitable, when dealing with massive datasets. Therefore, various asynchronous or distributed extensions have been proposed [16, 27, 18, 28, 29], where each term is handled independently by a processing unit and the convergence toward an aggregate solution to the optimization problem is ensured via a suitable communication strategy between those processing units. However, the convergence analysis of primal-dual distributed algorithms is usually based on fixed-point theory, that require specific probabilistic assumptions on the block update rule. Moreover, the integration of accelerations, such as preconditioning, into those distributed algorithms is difficult.

In this paper, we focus instead on another approach, namely the dual block preconditioned forward-backward algorithm that we recently proposed in [25], which can be viewed as a block-coordinate implementation of the dual ascent method. We propose here a distributed asynchronous version for the latter, by considering each involved function as locally related to a node of a connected hypergraph, where communications are allowed between neighboring nodes that share the same hyperedge. This leads to a novel scheme for computing the proximity operator of any sum of convex functions involving linear operators, which is well-suited to architectures involving multiple computing units. As its centralized counterpart [25], our method takes advantage of variable metric techniques that have been shown to be efficient for accelerating the convergence speed of proximal approaches [30, 31, 32]. It also benefits from the classical key advantage of proximal splitting strategies, namely its ability to handle a finite sum of convex functions without having to invert any of the involved linear operators. Furthermore, its convergence is guaranteed under mild assumptions on the node activation and synchronization rules.

The remainder of this paper is organized as follows: in Section 2 we recall some fundamental background and present the centralized dual block-coordinate

forward-backward algorithm from [25] for computing proximity operators. In Section 3, we introduce our novel asynchronous version for this algorithm, its convergence properties and a dimension reduction strategy for limiting communication cost within nodes. In Section 4, we discuss a useful special case of our algorithm for an important class of hypergraph structure and we describe its practical implementation on a distributed architecture. Section 4.3 shows the good performance of the proposed algorithm in the context of video denoising. Finally, some conclusions are drawn in Section 5.

2. Problem formulation

2.1. Optimization background

Let $\Gamma_0(\mathbb{R}^N)$ denote the class of proper lower-semicontinuous convex functions from $\mathbb{R}^N$ to $(-\infty, +\infty]$ and let $B \in \mathbb{R}^{N\times N}$ be a symmetric positive definite matrix. The proximity operator of $\psi \in \Gamma_0(\mathbb{R}^N)$ at $\widetilde{x} \in \mathbb{R}^N$ relative to the metric induced by B is denoted by $\mathrm{prox}_{B,\psi}(\widetilde{x})$ and defined as the unique solution to the following minimization problem [33, 1]:

$$\underset{x\in\mathbb{R}^N}{\text{minimize}} \;\; \psi(x) + \frac{1}{2}\|x-\widetilde{x}\|_B^2, \tag{1}$$

where the weighted norm $\|\cdot\|_B$ is defined as $\langle\cdot\,|\,B\,\cdot\rangle^{1/2}$ with $\langle\cdot\,|\,\cdot\rangle$ the usual scalar product of $\mathbb{R}^N$. When B is set to I_N, the identity matrix of $\mathbb{R}^N$, the standard proximity operator prox_{ψ} is recovered.
We now define the conjugate of a function $\psi \in \Gamma_0(\mathbb{R}^N)$ as

$$\psi^* : \mathbb{R}^N \to (-\infty, +\infty] : x \mapsto \sup_{v\in\mathbb{R}^N} \left(\langle v \mid x\rangle - \psi(v)\right). \tag{2}$$

Following Moreau's decomposition theorem [34],

$$\mathrm{prox}_{B,\psi^*} = I_N - B^{-1}\mathrm{prox}_{B^{-1},\psi}(B\,\cdot). \tag{3}$$

2.2. Minimization problem

This paper addresses the problem of computing the proximity operator of the following sum of functions at some given point $\widetilde{x}$ of $\mathbb{R}^N$:

$$\big(\forall x \in \mathbb{R}^N\big) \qquad G(x) = \sum_{j=1}^{J} g_j(A_j x), \tag{4}$$

where, for every $j \in \{1, \ldots, J\}$, $g_j : \mathbb{R}^{M_j} \to (-\infty, +\infty]$ is a proper lower-semicontinuous convex possibly nonsmooth function and A_j is a linear operator in $\mathbb{R}^{M_j \times N}$. In addition, it is assumed that $\cap_{j=1}^{J} \mathrm{dom}\,(g_j \circ A_j) \neq \varnothing$.
Computing the proximity operator of G amounts to finding the solution to the following minimization problem:

$$\text{Find } \widehat{x} = \mathrm{prox}_G(\widetilde{x}) = \underset{x \in \mathbb{R}^N}{\mathrm{argmin}} \ \sum_{j=1}^{J} g_j(A_j x) + \frac{1}{2}\|x - \widetilde{x}\|^2. \tag{5}$$

As we will see in Section 4.3, the latter problem also arises in the computation of the maximum a posteriori solution for the denoising problem that consists of recovering $\widehat{x}$ from a noisy observation $\widetilde{x}$ in the presence of an additive zero-mean white Gaussian noise and of a prior density $\exp(-G)$. [1].

Primal-dual algorithms [10, 14, 15, 16] amounts to solve Problem (5) by making use of its dual formulation given by:

$$\text{Find } \quad \widehat{y} = \underset{y=(y^j)_{1 \leqslant j \leqslant J} \in \mathbb{R}^M}{\mathrm{argmin}} \ \frac{1}{2}\Big\|\widetilde{x} - \sum_{j=1}^{J} A_j^\top y^j\Big\|^2 + \sum_{j=1}^{J} g_j^*(y^j), \tag{6}$$

where $M = \sum_{j=1}^{J} M_j$ and $(g_j^*)_{1 \leqslant j \leqslant J}$ are the Fenchel-Legendre conjugate functions of $(g_j)_{1 \leqslant j \leqslant J}$. Particularly efficient primal-dual approaches take advantage of the strongly convex term involved in the cost function in (5) [35, 36, 25]. In this work, we focus on the Dual Block Preconditioned Forward-Backward algorithm, recently proposed in [25] (see Algorithm 1).

Algorithm 1 benefits from the acceleration provided by variable metric meth-

Algorithm 1: Dual Block Preconditioned Forward-Backward

Initialization:

$B_j \in \mathbb{R}^{M_j \times M_j}$ with $B_j \succeq A_j A_j^\top, \quad \forall j \in \{1, \dots, J\}$

$\epsilon \in (0, 1]$, $(y_0^j)_{1 \leqslant j \leqslant J} \in \mathbb{R}^M$, $x_0 = \widetilde{x} - \sum_{j=1}^{J} A_j^\top y_0^j$.

Main loop:

for $n = 0, 1, \dots$ **do**

  $\gamma_n \in [\epsilon, 2 - \epsilon]$

  $j_n \in \{1, \dots, J\}$

  $\widetilde{y}_n^{j_n} = y_n^{j_n} + \gamma_n B_{j_n}^{-1} A_{j_n} x_n$

  $y_{n+1}^{j_n} = \widetilde{y}_n^{j_n} - \gamma_n B_{j_n}^{-1} \mathrm{prox}_{\gamma_n B_{j_n}^{-1}, g_{j_n}} \left(\gamma_n^{-1} B_{j_n} \widetilde{y}_n^{j_n} \right)$

  $y_{n+1}^j = y_n^j, \quad \forall j \in \{1, \dots, J\} \backslash \{j_n\}$

  $x_{n+1} = x_n - A_{j_n}^\top (y_{n+1}^{j_n} - y_n^{j_n})$.

end

ods through the introduction of preconditioning matrices $(B_j)_{1 \leqslant j \leqslant J}$. Note that a non-preconditioned version is obtained by setting $(\forall j \in \{1, \dots, J\})$ $B_j = \|A_j\|^2 I_{M_j}$ where $\|A_j\|$ denotes the spectral norm of A_j. Moreover, when at iteration $n \in \mathbb{N}$, all the dual variables $y_n^{j_n}$ with $j_n \in \{1, \dots, J\}$ are updated in a parallel manner followed by an update of the primal variable x_n, one recovers the Parallel Dual Forward-Backward proposed in [35]. Convergence guarantees on both generated primal sequence $(x_n)_{n \in \mathbb{N}}$ and dual sequences $(y_n^j)_{n \in \mathbb{N}*}$ with $j \in \{1, \dots, J\}$ have been established in [25] under a quasi-cyclic rule on the block selection (i.e., each block must be updated at least once every K iterations, with $K \geqslant J$). Furthermore, results in terms of practical convergence speed have revealed the effectiveness of the above algorithm compared to existing algorithms in the literature.

3. Proposed distributed algorithm

We ground our work on the previous algorithm in order to design a novel distributed (i.e., multi-node) solution to Problem (5). This can be achieved by resorting to a global consensus technique [37, 16, 27, 3, 38] and rewriting the

problem in the following form:

$$\text{Find } \widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}=(x^j)_{1\leqslant j\leqslant J}\in\Lambda}{\operatorname{argmin}} \sum_{j=1}^{J} g_j(A_j x^j) + \frac{1}{2}\sum_{j=1}^{J} \|x^j - \widetilde{x}\|^2_{\Omega_j}, \tag{7}$$

where $(\Omega_j)_{1\leqslant j\leqslant J}$ are diagonal $N\times N$ matrices with positive diagonal elements and Λ is the vector subspace of $\mathbb{R}^{NJ}$ defined so as to introduce suitable coupling constraints on the vectors $(x^j)_{1\leqslant j\leqslant J}$. The most standard choice for such constraint set is

$$\Lambda = \left\{ \begin{bmatrix} x^1 \\ \vdots \\ x^J \end{bmatrix} \in \mathbb{R}^{NJ} \mid x^1 = \ldots = x^J \right\}. \tag{8}$$

Provided that

$$\sum_{j=1}^{J} \Omega_j = I_N, \tag{9}$$

we notice that the solution to Problem (7) yields a vector in $\mathbb{R}^{NJ}$,for which the components $(x^j)_{1\leqslant j\leqslant J}$ are all equal, and identical to the solution $\widehat{x}$ to Problem (5).

3.1. Local form of consensus

Let us now split the constraint set Λ into L local linear constraints $(\Lambda_\ell)_{1\leqslant\ell\leqslant L}$. For every $\ell\in\{1,\ldots,L\}$, each constraint set Λ_ℓ handles a nonempty subset $\mathbb{V}_\ell$ of $\{1,\ldots,J\}$ with cardinality κ_ℓ such that, for every $\boldsymbol{x} = [(x^1)^\top,\ldots,(x^J)^\top]^\top \in \mathbb{R}^{NJ}$,

$$\boldsymbol{x}\in\Lambda \quad\Leftrightarrow\quad (\forall \ell\in\{1,\ldots,L\}) \quad (x^j)_{j\in\mathbb{V}_\ell}\in\Lambda_\ell. \tag{10}$$

Examples of vector subspaces $(\Lambda_\ell)_{1\leqslant\ell\leqslant L}$ allowing this condition to be satisfied are discussed in Section 3.3. Each node $j\in\{1,\ldots,J\}$ is associated with function g_j, which is considered local and processes its own private data. Moreover, each node j is allowed to communicate with nodes that belong to the same set $\mathbb{V}_\ell$. The sets $(\mathbb{V}_\ell)_{1\leqslant\ell\leqslant L}$ can thus be viewed as the hyperedges of a hypergraph with

J nodes. It is worth noticing that the case of a graph topology, considered for instance in the distributed projected subgradient method from [38], is a special case of this formulation when setting the cardinality of the set $\mathbb{V}_\ell$ to $\kappa_\ell = 2$ for every $\ell \in \{1, \dots, L\}$.

Figure 1 shows an illustrative example, where the hypergraph is composed of $J = 7$ nodes associated with functions $(g_j)_{1 \leqslant j \leqslant 7}$ and $L = 4$ hyperedges represented by the sets $(\mathbb{V}_\ell)_{1 \leqslant \ell \leqslant 4}$ with cardinalities $\kappa_1 = 3, \kappa_2 = 2, \kappa_3 = 2$, and $\kappa_4 = 3$, respectively. Node 4 belonging to the set $\mathbb{V}_2$ can communicate with node 5. Besides, node 3 belongs to $\mathbb{V}_1$ and $\mathbb{V}_4$, hence it is allowed to communicate with nodes $\{1, 2, 5, 7\}$.

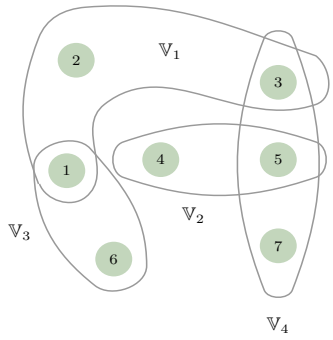


Figure 1: Connected hypergraph of $J = 7$ nodes and $L = 4$ hyperedges.

Let us define, for every $\ell \in \{1, \dots, L\}$, the matrix $\boldsymbol{S}_\ell \in \mathbb{R}^{N\kappa_\ell \times NJ}$ associated with constraint set Λ_ℓ, which extracts the vector $(x^j)_{j \in \mathbb{V}_\ell}$ from the concatenated vector $\boldsymbol{x} = [(x^1)^\top, \dots, (x^J)^\top]^\top \in \mathbb{R}^{NJ}$:

$$(x^j)_{j \in \mathbb{V}_\ell} = [(x^{\mathsf{i}(\ell,1)})^\top, \dots, (x^{\mathsf{i}(\ell,\kappa_\ell)})^\top]^\top = \boldsymbol{S}_\ell \, \boldsymbol{x}, \tag{11}$$

where $\mathsf{i}(\ell, 1), \dots, \mathsf{i}(\ell, \kappa_\ell)$ denote the elements of $\mathbb{V}_\ell$ ordered in an increasing manner. The transpose matrix of $(\boldsymbol{S}_\ell)_{1 \leqslant \ell \leqslant L}$ is such that, for every $v^\ell = (v^{\ell,k})_{1 \leqslant k \leqslant \kappa_\ell} \in \mathbb{R}^{N\kappa_\ell}$,

$$\boldsymbol{x} = [(x^1)^\top, \dots, (x^J)^\top] = \boldsymbol{S}_\ell^\top v^\ell, \tag{12}$$

where

$$x^j = \begin{cases} v^{\ell,k} & \text{if } j = \mathsf{i}(\ell,k) \text{ with } k \in \{1,\dots,\kappa_\ell\} \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

From a signal processing standpoint, the matrix $\boldsymbol{S}_\ell$ can be viewed as a decimation operator, while its transpose is the associated interpolator.

The above definitions allow us to propose the following equivalent formulation of Problem (7):

$$\text{Find } \widehat{\boldsymbol{x}} = \underset{\boldsymbol{x}=(x^j)_{1\leqslant j\leqslant J}\in\mathbb{R}^{NJ}}{\operatorname{argmin}} \ \sum_{j=1}^{J} g_j(A_j x^j) + \sum_{\ell=1}^{L} \iota_{\Lambda_\ell}(\boldsymbol{S}_\ell\,\boldsymbol{x}) + \frac{1}{2}\sum_{j=1}^{J} \|x^j - \widetilde{x}\|^2_{\Omega_j}. \tag{14}$$

The main difference between formulations (7) and (14) is the introduction of the term $\sum_{\ell=1}^{L} \iota_{\Lambda_\ell}(\boldsymbol{S}_\ell\,\boldsymbol{x})$, where ι_{Λ_ℓ} denotes the indicator function of the set Λ_ℓ, which is 0 for every $z \in \Lambda_\ell$, and $+\infty$ elsewhere.

This latter formulation makes the link with Problem (5) more explicit.

More precisely, in order to solve Problem (14) using Algorithm 1, it is necessary to set:

- $J' = J + L$,
- $(\forall \ell \in \{1,\dots,L\}) \quad M_{J+\ell} = N\kappa_\ell$,
- $M' = \sum_{j=1}^{J'} M_j$,
- $(\forall j \in \{1,\dots,J\}) \quad \boldsymbol{A}_j = [\ \underbrace{0\dots0}_{N(j-1)\times}\ A_j\Omega_j^{-1/2}\ \underbrace{0\dots0}_{N(J-j)\times}\ ]$,
- $\boldsymbol{D} = \operatorname{Diag}(\Omega_1^{-1/2},\dots,\Omega_J^{-1/2})$,
- $(\forall \ell \in \{1,\dots,L\})\ g_{J+\ell} = \iota_{\Lambda_\ell}$ and $\boldsymbol{A}_{J+\ell} = \boldsymbol{S}_\ell\boldsymbol{D}$.

Then, Problem (14) is recast in the following way:

$$\text{Find } \widehat{\boldsymbol{x}} = \boldsymbol{D}\widehat{\boldsymbol{x}}' \quad \text{such that}$$
$$\widehat{\boldsymbol{x}}' = \underset{\boldsymbol{x}' \in \mathbb{R}^{NJ}}{\operatorname{argmin}} \ \sum_{j=1}^{J'} g_j(\boldsymbol{A}_j \boldsymbol{x}') + \frac{1}{2}\|\boldsymbol{x}' - \widetilde{\boldsymbol{x}}'\|^2, \tag{15}$$

where $\widetilde{\boldsymbol{x}}' = [\Omega_1^{1/2}\widetilde{x}^\top, \ldots, \Omega_J^{1/2}\widetilde{x}^\top]^\top \in \mathbb{R}^{NJ}$.

3.2. Derivation of the proposed algorithm

The application of Algorithm 1 for the resolution of Problem (15) yields:

$$\left\lfloor \begin{array}{l} B_j \in \mathbb{R}^{M_j \times M_j} \text{ with } B_j \succeq \boldsymbol{A}_j \boldsymbol{A}_j^\top, \quad j \in \{1, \ldots, J'\} \\ \epsilon \in (0, 1] \\ (y_0^j)_{1 \leqslant j \leqslant J'} \in \mathbb{R}^{M'} \\ \boldsymbol{x}'_0 = \widetilde{\boldsymbol{x}}' - \sum_{j=1}^{J'} \boldsymbol{A}_j^\top y_0^j. \end{array} \right.$$
For $n = 0, 1, \ldots$
$$\left\lfloor \begin{array}{l} \gamma_n \in [\epsilon, 2 - \epsilon] \\ j_n \in \{1, \ldots, J'\} \\ \widetilde{y}_n^{j_n} = y_n^{j_n} + \gamma_n (B_{j_n})^{-1} \boldsymbol{A}_{j_n} \boldsymbol{x}'_n \\ y_{n+1}^{j_n} = \widetilde{y}_n^{j_n} - \gamma_n (B_{j_n})^{-1} \operatorname{prox}_{\gamma_n (B_{j_n})^{-1}, g_{j_n}} \left(\gamma_n^{-1} B_{j_n} \widetilde{y}_n^{j_n}\right) \\ y_{n+1}^{j} = y_n^{j}, \quad j \in \{1, \ldots, J'\} \backslash \{j_n\} \\ \boldsymbol{x}'_{n+1} = \boldsymbol{x}'_n - \boldsymbol{A}_{j_n}^\top (y_{n+1}^{j_n} - y_n^{j_n}). \end{array} \right. \tag{16}$$

The following convergence properties of the above algorithm can be deduced from [25, Prop. 1-2]:

Theorem 3.1. *Assume that*

(i) $(g_j)_{1 \leqslant j \leqslant J}$ *are semi-algebraic functions, and for every* $j \in \{1, ..., J\}$, *the restriction of* g_j^* *on its domain is continuous ;*

(ii) *the sequence* $(j_n)_{n \in \mathbb{N}}$ *follows a quasi-cyclic rule, i.e., there exists* $K \geqslant J'$ *such that, for every* $n \in \mathbb{N}$, $\{1, \ldots, J'\} \subset \{j_n, \ldots, j_{n+K-1}\}$.

Let $(\boldsymbol{x}'_n)_{n\in\mathbb{N}}$, $(\boldsymbol{y}_n = (y_n^j)_{1\leqslant j\leqslant J'})_{n\in\mathbb{N}}$ *be sequences generated by Algorithm* (16), *and* $(\boldsymbol{x}_n)_{n\in\mathbb{N}} = (\boldsymbol{D}\boldsymbol{x}'_n)_{n\in\mathbb{N}}$. *Then, if* $(\boldsymbol{y}_n)_{n\in\mathbb{N}}$ *is bounded, then* $(\boldsymbol{x}'_n)_{n\in\mathbb{N}}$ *converges to the solution* $\widehat{\boldsymbol{x}}'$ *to Problem* (15), *and* $(\boldsymbol{x}_n)_{n\in\mathbb{N}}$ *converges to the solution* $\widehat{\boldsymbol{x}}$ *to Problem* (7). *Moreover, there exists* $\alpha \in (0, +\infty)$ *such that*

$$\lim_{n\to+\infty} n^\alpha \|\boldsymbol{x}_n - \widehat{\boldsymbol{x}}\| \in \mathbb{R}. \tag{17}$$

We now show how the above algorithm can be simplified.
First, note that $(\forall j \in \{1,\dots,J\})$ $\boldsymbol{A}_j\boldsymbol{A}_j^\top = A_j\Omega_j^{-1}A_j^\top$ and $(\forall \ell \in \{1,\dots,L\})$ $\|\boldsymbol{S}_\ell\boldsymbol{D}\| = \max_{j\in\mathbb{V}_\ell} \|\Omega_j^{-1/2}\|$. It can also be observed that $(\forall \ell \in \{1,\dots,L\})$ $(\forall \gamma \in]0,+\infty[)$,

$$\operatorname{prox}_{\gamma^{-1}g_{J+\ell}}(\gamma^{-1}\cdot) = \gamma^{-1}\Pi_{\Lambda_\ell}, \tag{18}$$

where Π_{Λ_ℓ} is the linear projector onto the vector space Λ_ℓ.
Hence, by setting

$$(\forall \ell \in \{1,\dots,L\}) \qquad B_{J+\ell} = \vartheta_\ell^{-1} I_{N\kappa_\ell} \tag{19}$$

with $\vartheta_\ell = \min_{j\in\mathbb{V}_\ell} \|\Omega_j\|$, and $(\forall j \in \{1,\dots,J\})$

$$\mathbb{V}_j^* = \big\{(\ell,k) \;\big|\; \ell \in \{1,\dots,L\},\, k \in \{1,\dots,\kappa_\ell\} \text{ and } \mathsf{i}(\ell,k) = j\big\}, \tag{20}$$

Algorithm (16) can be re-expressed as

$$
\begin{array}{l}
B_j \in \mathbb{R}^{M_j \times M_j} \text{ with } B_j \succeq A_j \Omega_j^{-1} A_j^\top, \quad j \in \{1, \dots, J\} \\
\vartheta_\ell = \min\limits_{j \in \mathbb{V}_\ell} \|\Omega_j\|, \quad \ell \in \{1, \dots, L\} \\
\epsilon \in (0, 1] \\
z_0^\ell \in \mathbb{R}^{N \kappa_\ell}, \quad \ell \in \{1, \dots, L\} \\
y_0^j \in \mathbb{R}^{M_j}, \quad j \in \{1, \dots, J\} \\
x_0^j = \widetilde{x} - \Omega_j^{-1} \Big(A_j^\top y_0^j + \sum_{(\ell,k) \in \mathbb{V}_j^*} z_0^{\ell,k} \Big), \quad j \in \{1, \dots, J\}.
\end{array}
$$

For $n = 0, 1, \dots$

$$
\begin{array}{l}
\gamma_n \in [\epsilon, 2 - \epsilon] \\
j_n \in \{1, \dots, J + L\} \\
\text{If } j_n \leqslant J \\
\quad \left| \begin{array}{l}
\widetilde{y}_n^{j_n} = y_n^{j_n} + \gamma_n (B_{j_n})^{-1} A_{j_n} x_n^{j_n} \\
y_{n+1}^{j_n} = \widetilde{y}_n^{j_n} - \gamma_n (B_{j_n})^{-1} \mathrm{prox}_{\gamma_n (B_{j_n})^{-1}, g_{j_n}} \big(\gamma_n^{-1} B_{j_n} \widetilde{y}_n^{j_n} \big) \\
y_{n+1}^j = y_n^j, \quad j \in \{1, \dots, J\} \backslash \{j_n\} \\
z_{n+1}^\ell = z_n^\ell, \quad \ell \in \{1, \dots, L\} \\
x_{n+1}^{j_n} = x_n^{j_n} - \Omega_{j_n}^{-1} A_{j_n}^\top (y_{n+1}^{j_n} - y_n^{j_n}) \\
x_{n+1}^j = x_n^j, \quad j \in \{1, \dots, J\} \backslash \{j_n\}
\end{array} \right. \\
\quad \text{else} \\
\quad \left| \begin{array}{l}
\ell_n = j_n - J \\
\widetilde{z}_n^{\ell_n} = z_n^{\ell_n} + \gamma_n \vartheta_{\ell_n} (x_n^j)_{j \in \mathbb{V}_{\ell_n}} \\
z_{n+1}^{\ell_n} = \widetilde{z}_n^{\ell_n} - \Pi_{\Lambda_{\ell_n}} (\widetilde{z}_n^{\ell_n}) \\
z_{n+1}^\ell = z_n^\ell, \quad \ell \in \{1, \dots, L\} \backslash \{\ell_n\} \\
y_{n+1}^j = y_n^j, \quad j \in \{1, \dots, J\} \\
\text{For } k = 1, \dots, \kappa_{\ell_n} \\
\quad \left| \; x_{n+1}^{\mathsf{i}(\ell_n,k)} = x_n^{\mathsf{i}(\ell_n,k)} - \Omega_{\mathsf{i}(\ell_n,k)}^{-1} (z_{n+1}^{\ell_n,k} - z_n^{\ell_n,k}) \right. \\
x_{n+1}^j = x_n^j, \quad j \notin \mathbb{V}_{\ell_n}.
\end{array} \right.
\end{array}
\tag{21}
$$

In this algorithm, for increased readibility, we have set, for every $n \in \mathbb{N}$,

$$\boldsymbol{x}_n = [(x_n^1)^\top, \dots, (x_n^J)^\top]^\top = \boldsymbol{D}\boldsymbol{x}'_n, \tag{22}$$

$$z_n^\ell = y_n^{J+\ell} \in \mathbb{R}^{N\kappa_\ell}, \quad \widetilde{z}_n^\ell = \widetilde{y}_n^{J+\ell} \in \mathbb{R}^{N\kappa_\ell}. \tag{23}$$

Furthermore, it can be noticed that, for every $n \in \mathbb{N}$ such that $j_n = J + \ell_n > J$,

$$\begin{aligned} \Pi_{\Lambda_{\ell_n}}(z_{n+1}^{\ell_n}) &= \Pi_{\Lambda_{\ell_n}}\big(\widetilde{z}_n^{\ell_n} - \Pi_{\Lambda_{\ell_n}}(\widetilde{z}_n^{\ell_n})\big) \\ &= \Pi_{\Lambda_{\ell_n}}\big(\widetilde{z}_n^{\ell_n}\big) - \Pi_{\Lambda_{\ell_n}}\big(\Pi_{\Lambda_{\ell_n}}(\widetilde{z}_n^{\ell_n})\big) \\ &= 0. \end{aligned} \tag{24}$$

Since, for every $\ell \in \{1, \dots, L\} \backslash \{\ell_n\}$, $z_{n+1}^\ell = z_n^\ell$, the latter equality can be extended by induction to

$$(\forall n \in \mathbb{N})(\forall \ell \in \{1, \dots, L\}) \quad \Pi_{\Lambda_\ell}(z_n^\ell) = 0, \tag{25}$$

using an appropriate initialization of the algorithm (e.g., by choosing $(\forall \ell \in \{1, \dots, L\})\ z_0^\ell = 0$). Hence, for every $n \in \mathbb{N}$ such that $j_n = J + \ell_n > J$,

$$\Pi_{\Lambda_{\ell_n}}(\widetilde{z}_n^\ell) = \gamma_n \vartheta_{\ell_n} \Pi_{\Lambda_{\ell_n}}\big((x_n^j)_{j \in \mathbb{V}_{\ell_n}}\big), \tag{26}$$

which implies that

$$z_{n+1}^{\ell_n} - z_n^{\ell_n} = \gamma_n \vartheta_{\ell_n}\big((x_n^j)_{j \in \mathbb{V}_{\ell_n}} - \Pi_{\Lambda_{\ell_n}}\big((x_n^j)_{j \in \mathbb{V}_{\ell_n}}\big)\big). \tag{27}$$

The second part of iteration n of (21) dealing with the case when $j_n > J$ can then be re-expressed as shown in the projection step of Algorithm 2 (lines 20 to 26). In the resulting algorithm, we were able to drop the variables $(z_n^\ell)_{1 \leqslant \ell \leqslant L}$, for every $n \in \mathbb{N}$.

The body of our proposed Algorithm 2 is composed of two main parts:

Algorithm 2: Distributed Preconditioned Dual Forward-Backward

Initialization:

$\mathbb{V}_\ell \equiv$ index set of nodes in hyperedge $\ell \in \{1, \dots, L\}$

$B_j \in \mathbb{R}^{M_j \times M_j}$ with $B_j \succeq A_j \Omega_j^{-1} A_j^\top, \quad j \in \{1, \dots, J\}$

$\vartheta_\ell = \min_{j \in \mathbb{V}_\ell} \|\Omega_j\|, \quad \ell \in \{1, \dots, L\}$

$\epsilon \in (0, 1]$

$y_0^j \in \mathbb{R}^{M_j}, x_0^j = \widetilde{x} - \Omega_j^{-1} A_j^\top y_0^j, \quad j \in \{1, \dots, J\}.$

Main loop:

for $n = 0, 1, \dots$ **do**

  $\gamma_n \in [\epsilon, 2 - \epsilon]$

  $j_n \in \{1, \dots, J + L\}$

  if $j_n \leqslant J$ **then**

    Local optimization:

    $\widetilde{y}_n^{j_n} = y_n^{j_n} + \gamma_n (B_{j_n})^{-1} A_{j_n} x_n^{j_n}$

    $y_{n+1}^{j_n} = \widetilde{y}_n^{j_n} - \gamma_n (B_{j_n})^{-1} \mathrm{prox}_{\gamma_n (B_{j_n})^{-1}, g_{j_n}} \left(\gamma_n^{-1} B_{j_n} \widetilde{y}_n^{j_n} \right)$

    $y_{n+1}^j = y_n^j, \quad j \in \{1, \dots, J\} \backslash \{j_n\}$

    $x_{n+1}^{j_n} = x_n^{j_n} - \Omega_{j_n}^{-1} A_{j_n}^\top (y_{n+1}^{j_n} - y_n^{j_n})$

    $x_{n+1}^j = x_n^j, \quad j \in \{1, \dots, J\} \backslash \{j_n\}$

  else

    Projection:

    $\ell_n = j_n - J$

    $y_{n+1}^j = y_n^j, \quad j \in \{1, \dots, J\}$

    $p_n^{\ell_n} = \Pi_{\Lambda_{\ell_n}} \left((x_n^j)_{j \in \mathbb{V}_{\ell_n}} \right)$

    for $k = 1, \dots, \kappa_{\ell_n}$ **do**

      $x_{n+1}^{\mathsf{i}(\ell_n, k)} = x_n^{\mathsf{i}(\ell_n, k)} + \gamma_n \vartheta_{\ell_n} \Omega_{\mathsf{i}(\ell_n, k)}^{-1} (p_n^{\ell_n, k} - x_n^{\mathsf{i}(\ell_n, k)})$

    end

    $x_{n+1}^j = x_n^j, \quad j \notin \mathbb{V}_{\ell_n}.$

  end

end

- A first local optimization step (lines 13 to 17) which is reminiscent of the Dual Block Forward-Backward algorithm where, at each iteration, a block j_n is selected and the associated dual and primal variables $y_n^{j_n}$ (line 14) and $x_n^{j_n}$ (line 16) are updated, respectively. Note that a fundamental difference between the proposed algorithm and Algorithm 1 lies in the fact that each block j_n is now associated with a local primal variable $x_n^{j_n}$ whereas, in Algorithm 1, x_n was a shared variable.

- A projection step (lines 20 to 26) in which a set $\mathbb{V}_{\ell_n}$ is selected and all the variables $(x^{j_n})_{j_n \in \mathbb{V}_{\ell_n}}$ are updated by means of a projection operating over the selected set $\mathbb{V}_{\ell_n}$.

In Algorithm 2, all computation steps only involve local variables, which is suitable for parallel processing. A high degree of flexibility is allowed in the quasi-cyclic rule for choosing the indices j_n and ℓ_n at each iteration n. The distributed Algorithm 2 inherits all the advantages of primal-dual methods, in particular it requires no inversion of matrices $(A_j)_{1 \leqslant j \leqslant J}$, which is critical when these matrices have a complex structure and are of very large size. Note that the proposed approach is quite different from the ones developed in [27, 28, 38] since it does not assume a random sweeping rule for the block index selection. Moreover, in contrast with the aforementioned works, its convergence analysis, secured by Theorem 3.1, does not rely on the nonexpansiveness property of the involved operators.

3.3. Consensus choice

We now discuss practical settings for the vector spaces $(\Lambda_\ell)_{1 \leqslant \ell \leqslant L}$ and the weights and the weight matrices $(\Omega_j)_{1 \leqslant j \leqslant J}$, that parameterize our consensus formulation (14). These choices are of main importance to devise efficient distributed schemes with limited communication cost and good practical convergence speed.

3.3.1. Generic case

When the operators $(A_j)_{1\leqslant j\leqslant J}$ have no specific structure, a natural choice for the vector spaces $(\Lambda_\ell)_{1\leqslant \ell\leqslant L}$ is to adopt a form similar to that of Λ in (8):

$$(\forall \ell \in \{1,\dots,L\}) \qquad \Lambda_\ell = \left\{ \begin{bmatrix} v^{\ell,1} \\ \vdots \\ v^{\ell,\kappa_\ell} \end{bmatrix} \in \mathbb{R}^{N\kappa_\ell} \mid v^{\ell,1} = \ldots = v^{\ell,\kappa_\ell} \right\}. \quad (28)$$

Note that (8), (10) and (28) imply that the hypergraph induced by the hyperedges $(\mathbb{V}_\ell)_{1\leqslant \ell\leqslant L}$ is connected (Figure 1 is an example of such a connected hypergraph). In this context, the connectivity of the hypergraph is essential in order to allow the global consensus solution to be reached.

For every $\ell \in \{1,\dots,L\}$, the projection onto Λ_ℓ is then simply expressed as

$$\big(\forall (v^{\ell,k})_{1\leqslant k\leqslant \kappa_\ell} \in \mathbb{R}^{N\kappa_\ell}\big) \qquad \Pi_{\Lambda_\ell}\big((v^{\ell,k})_{1\leqslant k\leqslant \kappa_\ell}\big) = [(\overline{v}^\ell)^\top,\dots,(\overline{v}^\ell)^\top]^\top, \quad (29)$$

where

$$\overline{v}^\ell = \text{mean}\left((v^{\ell,k})_{1\leqslant k\leqslant \kappa_\ell}\right) \quad (30)$$

and mean$(\cdot)$ designates the arithmetic mean operation (i.e. mean $\left((v^{\ell,k})_{1\leqslant k\leqslant \kappa_\ell}\right) = \kappa_\ell^{-1}\sum_{k=1}^{\kappa_\ell} v^{\ell,k}$). In addition, Condition (9) is met by simply choosing $(\forall j \in \{1,\dots,J\})\ \Omega_j = \omega_j I_N$, where $(\omega_j)_{1\leqslant j\leqslant J} \in (0,1]^J$ are such that $\sum_{j=1}^J \omega_j = 1$. These simplifications lead to the following modifications of lines 22-25 in Algorithm 2:

$$\begin{array}{l} \overline{x}_n^{\ell_n} = \text{mean}\left((x_n^j)_{j\in\mathbb{V}_{\ell_n}}\right) \\ \text{For } k = 1,\dots,\kappa_{\ell_n} \\ \left\lfloor\ x_{n+1}^{\mathrm{i}(\ell_n,k)} = x_n^{\mathrm{i}(\ell_n,k)} + \gamma_n \vartheta_{\ell_n} \omega_{\mathrm{i}(\ell_n,k)}^{-1} (\overline{x}_n^{\ell_n} - x_n^{\mathrm{i}(\ell_n,k)}). \end{array} \quad (31)$$

3.3.2. Dimension reduction

Under its previous form, Algorithm 2 requires each node of the hypergraph to handle a local copy of several variables. In particular, for the j-th node, a vec-

tor x_n^j of dimension N needs to be stored, which may be prohibitive for highly dimensional problems. Fortunately, very often in signal and image processing applications, the operators $(A_j)_{1\leqslant j\leqslant J}$ have a sparse block structure, which mitigates this problem. More specifically, it will be assumed subsequently that

$$(\forall j \in \{1,\dots,J\})\big(\forall x^j = ([x^j]_t)_{1\leqslant t\leqslant T} \in \mathbb{R}^N\big) \qquad A_j x^j = \sum_{t\in\mathbb{T}_j} \mathcal{A}_{j,t}[x^j]_t \quad (32)$$

where, for every $j \in \{1,\dots,J\}$, $[x^j]_t$ is a vector corresponding to a block of data of dimension L, T is the overall number of blocks (i.e., $N = TL$), and $\mathbb{T}_j \subset \{1,\dots,T\}$ defines the reduced index subset of the components of vector x^j acting on the operator A_j. In the above equation, $(\mathcal{A}_{j,t})_{t\in\mathbb{T}_j}$ are the associated reduced-size matrices of dimensions $M_j \times L$. Similarly to the way x^j has been block-decomposed, we split the diagonal matrix Ω_j as $\Omega_j = \mathrm{Diag}\,(\Omega_{j,1},\dots,\Omega_{j,T})$ where, for every $t \in \{1,\dots,T\}$, $\Omega_{j,t}$ is a diagonal matrix of size $L \times L$. It then obviously holds that $A_j\Omega_j^{-1}A_j^\top = \sum_{t\in\mathbb{T}_j} \mathcal{A}_{j,t}\Omega_{j,t}^{-1}\mathcal{A}_{j,t}^\top$. To avoid degenerate cases, we will subsequently assume that $(\forall j \in \{1,\dots,J\})\ \mathbb{T}_j \neq \varnothing$ and $\bigcup_{j=1}^{J} \mathbb{T}_j = \{1,\dots,T\}$.

In our distributed formulation, the specific form of the operators $(A_j)_{1\leqslant j\leqslant J}$ suggests to set the vector subspaces $(\Lambda_\ell)_{1\leqslant \ell\leqslant L}$ so as to reach the consensus only for the components $([x^j]_t)_{1\leqslant j\leqslant J, t\in\mathbb{T}_j}$ of vectors $(x^j)_{1\leqslant j\leqslant J}$. This means that the space Λ (resp. Λ_ℓ with $\ell \in \{1,\dots,L\}$) is defined as

$$(x^j)_{1\leqslant j\leqslant J} \in \Lambda \;\Leftrightarrow\; (\forall (j,j') \in \{1,\dots,J\}^2)(\forall t \in \mathbb{T}_j \cap \mathbb{T}_{j'}) \quad [x^j]_t = [x^{j'}]_t \quad (33)$$

$$\text{(resp. } (x^j)_{j\in\mathbb{V}_\ell} \in \Lambda_\ell \;\Leftrightarrow\; (\forall (j,j') \in \mathbb{V}_\ell^2)(\forall t \in \mathbb{T}_j \cap \mathbb{T}_{j'}) \quad [x^j]_t = [x^{j'}]_t).$$

It can be noticed that, although the hypergraph must still be built so that (10) holds, Λ is no longer given by (8), since the components $([x^j]_t)_{1\leqslant j\leqslant J, t\notin\mathbb{T}_j}$ are unconstrained. The main advantage of this choice is that Problem (7) then

decouples into two optimization problems:

- the minimization of the function

$$([x^j]_t)_{1\leqslant j\leqslant J, t\in\mathbb{T}_j} \mapsto \sum_{j=1}^{J} g_j\Big(\sum_{t\in\mathbb{T}_j} \mathcal{A}_{j,t}[x^j]_t\Big) + \frac{1}{2}\sum_{j=1}^{J}\sum_{t\in\mathbb{T}_j} \|[x^j]_t - [\widetilde{x}]_t\|^2_{\Omega_{j,t}} \tag{34}$$

  subject to Constraint (33);

- the unconstrained minimization of the function

$$([x^j]_t)_{1\leqslant j\leqslant J, t\notin\mathbb{T}_j} \mapsto \sum_{j=1}^{J}\sum_{t\notin\mathbb{T}_j} \|[x^j]_t - [\widetilde{x}]_t\|^2_{\Omega_{j,t}}. \tag{35}$$

Since the second problem is trivial, the variables $([x_n^j]_t)_{1\leqslant j\leqslant J, t\notin\mathbb{T}_j}$ generated at each iteration $n \in \mathbb{N}$ of Algorithm 2 are useless and, consequently, they can be discarded. By doing so, only the $|\mathbb{T}_j|$ vectors[1] $([x_n^j]_t)_{t\in\mathbb{T}_j}$ of dimension L need to be stored at the j-th node (instead of T vectors of this size) and the number of computations to be performed during the projection step is also sharply diminished.

This yields Algorithm 3 where, in the synchronization step, averaging operations corresponding to the projection onto Λ_{ℓ_n} have been substituted for lines 22-25 in Algorithm 2. The notation

$$(\forall t \in \{1,\dots,T\}) \qquad \mathbb{T}_t^* = \big\{j \in \{1,\dots,J\} \;\big|\; t \in \mathbb{T}_j\big\}, \tag{36}$$

has been introduced for the computation of the averages. In particular, in line 29 of Algorithm 3, if $\mathbb{V}_\ell \cap \mathbb{T}_t^*$ is a singleton, which means that the t-th block component of the vector x appears only once in the expression of $g_j(A_j x)$ for indices j in the ℓ_n-th hyperedge, then the averaging reduces to setting $[x_{n+1}^j]_t = [x_n^j]_t$. It is also worthwhile to note that, when $(\forall j \in \{1,\dots,J\})\ \mathbb{T}_j = \{1,\dots,T\}$, the

[1] $|S|$ is the cardinality of a set S.

consensus solution described in Section 3.3.1 is recovered. It must be nonetheless pointed out that, in general, to ensure the equivalence between the minimization of (34) subject to Constraint (33) and the resolution of Problem (5), the following condition has to be substituted for (9):

$$(\forall t \in \{1, \dots, T\}) \qquad \sum_{j \in \mathbb{T}_t^*} \Omega_{j,t} = I_L. \tag{37}$$

In Algorithm 3, this was simply achieved by setting $(\forall j \in \{1, \dots, J\})$ $(\forall t \in \mathbb{T}_j)$ $\Omega_{j,t} = \omega_{j,t} I_L$, where $(\omega_{j,t})_{1 \leqslant j \leqslant J, t \in \mathbb{T}_j}$ are positive real such that $(\forall t \in \{1, \dots, T\})$ $\sum_{j \in \mathbb{T}_t^*} \omega_{j,t} = 1$. In turn, the notation $(\Omega_{j,t})_{1 \leqslant j \leqslant J, t \notin \mathbb{T}_j}$ is no longer used in this algorithm.

Algorithm 3 can give rise to a variety of distributed implementations. In the remainder of the paper, we will focus on a useful, simpler, specific instance of this algorithm.

4. A useful special case

Let us consider the case when $C \leqslant J$ processing units are available. To simplify our presentation, we will restrict our attention to a case of practical interest, that arises for example in the video denoising application described in Section 4.3, by making the following assumptions.

Assumption 4.1.

(i) *The hyperedges* $(\mathbb{V}_\ell)_{1 \leqslant \ell \leqslant C}$ *form a partition of* $\{1, \dots, J\}$.

(ii) *For every* $\ell \in \{1, \dots, C\}$, *let* $\mathbb{T}_{\mathbb{V}_\ell} = \bigcup_{j \in \mathbb{V}_\ell} \mathbb{T}_j$.

  (a) *For every* $(\ell, \ell') \in \{1, \dots, C\}^2$, $\mathbb{T}_{\mathbb{V}_\ell} \cap \mathbb{T}_{\mathbb{V}_{\ell'}} = \varnothing$ *if* $|\ell - \ell'| > 1$.

  (b) *For every* $\ell \in \{2, \dots, C-1\}$, $\mathbb{T}_{\mathbb{V}_{\ell-1}} \cap \mathbb{T}_{\mathbb{V}_\ell} \cap \mathbb{T}_{\mathbb{V}_{\ell+1}} = \varnothing$.

An example of hypergraph satisfying Assumption 4.1(i) is displayed in Figure 2. For every $\ell \in \{1, \dots, C\}$, $\mathbb{T}_{\mathbb{V}_\ell}$ is the set of the block indices t of the components

Algorithm 3: Distributed Preconditioned Dual Forward-Backward after Dimension Reduction

Initialization:

$\mathbb{V}_\ell \equiv$ index set of nodes in hyperedge $\ell \in \{1,\dots,L\}$

$\mathbb{T}_j \equiv$ index set of blocks used at node $j \in \{1,\dots,J\}$

$\mathbb{T}_t^* \equiv$ index set of nodes using block $t \in \{1,\dots,T\}$

$\{\omega_{j,t} \mid 1 \leqslant j \leqslant J, t \in \mathbb{T}_j\} \subset (0,1]$ such that $(\forall t \in \{1,\dots,T\}) \sum_{j \in \mathbb{T}_t^*} \omega_{j,t} = 1$

$B_j \in \mathbb{R}^{M_j \times M_j}$ with $B_j \succeq \sum_{t \in \mathbb{T}_j} \omega_{j,t}^{-1} \mathcal{A}_{j,t} \mathcal{A}_{j,t}^\top, \quad j \in \{1,\dots,J\}$

$\vartheta_\ell = \min_{j \in \mathbb{V}_\ell, t \in \mathbb{T}_j} \omega_{j,t}, \quad \ell \in \{1,\dots,L\}$

$\epsilon \in (0,1]$

$y_0^j \in \mathbb{R}^{M_j}, [x_0^j]_t = [\widetilde{x}]_t - \omega_{j,t}^{-1} \mathcal{A}_{j,t}^\top y_0^j, \quad j \in \{1,\dots,J\}, t \in \mathbb{T}_j.$

Main loop:

for $n = 0, 1, \dots$ **do**

  $\gamma_n \in [\epsilon, 2-\epsilon]$

  $j_n \in \{1,\dots,J+L\}$

  if $j_n \leqslant J$ **then**

    Local optimization:

    $\widetilde{y}_n^{j_n} = y_n^{j_n} + \gamma_n B_{j_n}^{-1} \sum_{t \in \mathbb{T}_{j_n}} \mathcal{A}_{j_n,t} [x_n^{j_n}]_t$

    $y_{n+1}^{j_n} = \widetilde{y}_n^{j_n} - \gamma_n B_{j_n}^{-1} \mathrm{prox}_{\gamma_n B_{j_n}^{-1}, g_{j_n}} \left(\gamma_n^{-1} B_{j_n} \widetilde{y}_n^{j_n}\right)$

    $y_{n+1}^j = y_n^j, \quad j \in \{1,\dots,J\} \backslash \{j_n\}$

    for $t \in \mathbb{T}_{j_n}$ **do**

      $[x_{n+1}^{j_n}]_t = [x_n^{j_n}]_t - \omega_{j_n,t}^{-1} \mathcal{A}_{j_n,t}^\top (y_{n+1}^{j_n} - y_n^{j_n})$

    end

    $([x_{n+1}^j]_t)_{t \in \mathbb{T}_j} = ([x_n^j]_t)_{t \in \mathbb{T}_j}, \quad j \in \{1,\dots,J\} \backslash \{j_n\}$

  else

    Projection:

    $\ell_n = j_n - J$

    $y_{n+1}^j = y_n^j, \quad j \in \{1,\dots,J\}$

    for $j \in \mathbb{V}_{\ell_n}$ **do**

      for $t \in \mathbb{T}_j$ **do**

        $[x_{n+1}^j]_t = [x_n^j]_t + \gamma_n \vartheta_{\ell_n} \omega_{j,t}^{-1} \left(\mathrm{mean}\left(([x_n^{j'}]_t)_{j' \in \mathbb{V}_{\ell_n} \cap \mathbb{T}_t^*} \right) - [x_n^j]_t \right)$

      end

    end

    $([x_{n+1}^j]_t)_{t \in \mathbb{T}_j} = ([x_n^j]_t)_{t \in \mathbb{T}_j}, \quad j \notin \mathbb{V}_{\ell_n}.$

  end

end

$[x^j]_t$ where j is any node in $\mathbb{V}_\ell$. According to Assumption 4.1-(ii)(a), these indices may only be common to hyperedges having preceding or following index values (i.e. $\ell - 1$ or $\ell + 1$). Finally, Assumption 4.1-(ii)(b) means that no intersection is allowed between block indices shared with the preceding and the succeeding hyperedges.

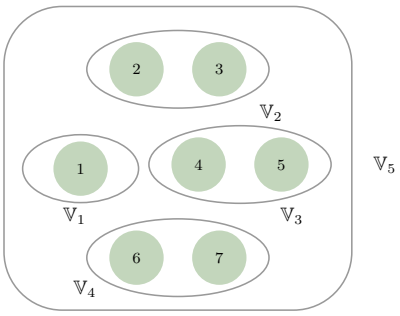


Figure 2: Hypergraph of $J = 7$ nodes, $C = 4$ computing units and $L = 5$ hyperedges.

4.1. Form of the algorithm

A practical instance of Algorithm 3 is then obtained by setting $L = C+1$ and by assuming that each hyperedge $\mathbb{V}_\ell$ with $\ell \in \{1, \ldots, C\}$ corresponds to a given computing unit where the computations are locally synchronized. In addition, hyperedge $\mathbb{V}_L$ is set to $\{1, \ldots, J\}$ in order to model global synchronization steps consisting of an averaging over all the available nodes. An example of such a structure is illustrated in Figure. 3. At each iteration n, only a subset $\mathbb{J}_{n,\ell}$ of dual variable indices is activated within the ℓ-th hyperedge. Their update is followed by either a possible local synchronization or a global one.

Algorithm 4 summarizes the proposed approach. For simplicity, the index L has been dropped in variable ϑ_L. Note that, if the local synchronization step is omitted (by setting $[x^j_{n+1}]_t = [x^j_{n+1/2}]_t$ in line 29), the algorithm still makes sense since it can be easily shown that it actually corresponds to a rewriting of Algorithm 3 in the case when $L = 1$ and $\mathbb{V}_1 = \{1, \ldots, J\}$. Note that in the case of Algorithm 4, the global synchronization step is mandatory although it does not neeed to be performed at each iteration, but only in a quasi-cyclic manner.

Conversely, this step may never be requested in Algorithm 3.

It should be emphasized that even in the case when all the dual variables are updated iteratively (i.e., $(\forall \ell \in \{1, \ldots, L\})\ (\forall n \in \mathbb{N})\ \mathbb{J}_{n,\ell} = \mathbb{V}_\ell$), Algorithm 4 exhibits a different structure from that of the parallel dual forward-backward algorithm in [35].

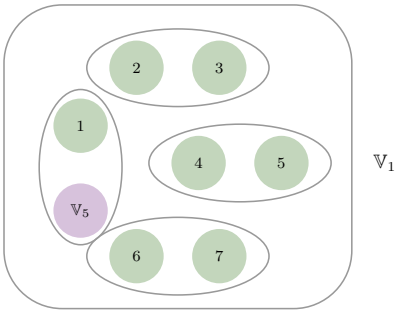


Figure 3: Hypergraph of $J = 8$ nodes including a fictitious node in charge of global synchronization, $C = 4$ computing units and $L = 1$ hyperedge.

4.2. Distributed implementation

We now look more precisely at the implementation of Algorithm 4 on a distributed architecture with $C \in \mathbb{N}^*$ computing units, each computing unit being indexed by an integer $c \in \{1, \ldots, C\}$. Figure 4 (top) shows an illustrative example of $C = 4$ computing units based on the hypergraph defined in Figure 2.

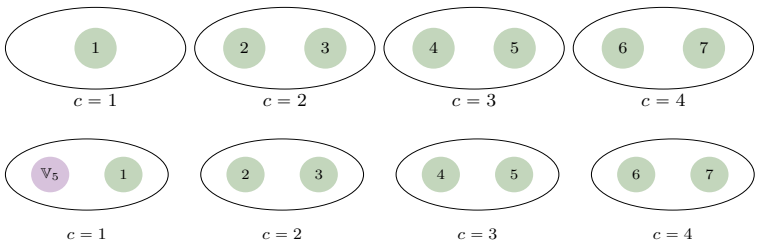


Figure 4: Example of partioning on $C = 4$ computing units, for the hypergraphs of Fig. 2 (top) and Fig. 3 (bottom), respectively.

Algorithm 4: Special case of distributed Preconditioned Dual Forward-Backward

Initialization:

$\mathbb{V}_\ell \equiv$ index set of nodes associated with computing unit $\ell \in \{1,\dots,C\}$

$\mathbb{T}_j \equiv$ index set of blocks used at node $j \in \{1,\dots,J\}$

$\mathbb{T}_t^* \equiv$ index set of nodes using block $t \in \{1,\dots,T\}$

$\{\omega_{j,t} \mid 1 \leqslant j \leqslant J, t \in \mathbb{T}_j\} \subset (0,1]$ such that $(\forall t \in \{1,\dots,T\}) \ \sum_{j \in \mathbb{T}_t^*} \omega_{j,t} = 1$

$B_j \in \mathbb{R}^{M_j \times M_j}$ with $B_j \geq \sum_{t \in \mathbb{T}_j} \omega_{j,t}^{-1} \mathcal{A}_{j,t} \mathcal{A}_{j,t}^\top, \quad j \in \{1,\dots,J\}$

$\vartheta = \min_{1 \leqslant j \leqslant J, 1 \leqslant t \leqslant T} \omega_{j,t}, \quad \vartheta_\ell = \min_{j \in \mathbb{V}_\ell, t \in \mathbb{T}_j} \omega_{j,t}, \ \ \ell \in \{1,\dots,C\}$

$\epsilon \in (0,1]$

$y_0^j \in \mathbb{R}^{M_j}, [x_0^j]_t = [\widetilde{x}]_t - \omega_{j,t}^{-1} \mathcal{A}_{j,t}^\top y_0^j, \quad j \in \{1,\dots,J\}, t \in \mathbb{T}_j.$

Main loop:

for $n = 0, 1, \dots$ **do**

  for $\ell = 1, \dots, C$ **do**

    $\mathbb{J}_{n,\ell} \subset \mathbb{V}_\ell$

    for $j \in \mathbb{J}_{n,\ell}$ **do**

      Local optimization:

      $\widetilde{y}_n^j = y_n^j + \gamma_n B_j^{-1} \sum_{t \in \mathbb{T}_j} \mathcal{A}_{j,t} [x_n^j]_t$

      $y_{n+1}^j = \widetilde{y}_n^j - \gamma_n B_j^{-1} \mathrm{prox}_{\gamma_n B_j^{-1}, g_j}\big(\gamma_n^{-1} B_j \widetilde{y}_n^j\big)$

      for $t \in \mathbb{T}_j$ **do**

        $[x_{n+1/2}^j]_t = [x_n^j]_t - \omega_{j,t}^{-1} \mathcal{A}_{j,t}^\top (y_{n+1}^j - y_n^j)$

      end

    end

    for $j \in \mathbb{V}_\ell \backslash \mathbb{J}_{n,\ell}$ **do**

      $y_{n+1}^j = y_n^j$

      $([x_{n+1/2}^j]_t)_{t \in \mathbb{T}_j} = ([x_n^j]_t)_{t \in \mathbb{T}_j}$

    end

    if *local synchronization is requested* **then**

      for $j \in \mathbb{V}_\ell$ **do**

        for $t \in \mathbb{T}_j$ **do**

          $[x_{n+1}^j]_t = [x_{n+1/2}^j]_t + \gamma_n \vartheta_\ell \omega_{j,t}^{-1} \big(\mathrm{mean}\,\big(([x_{n+1/2}^{j'}]_t)_{j' \in \mathbb{V}_\ell \cap \mathbb{T}_t^*}\big) - [x_{n+1/2}^j]_t \big)$

        end

      end

    end

  end

  if *global synchronization is requested* **then**

    for $t = 1, \dots, T$ **do** $[\overline{x}_n]_t = \mathrm{mean}\,\big(([x_{n+1/2}^j]_t)_{j \in \mathbb{T}_t^*}\big)$;

    for $j = 1, \dots, J$ **do**

      for $t \in \mathbb{T}_j$ **do**

        $[x_{n+1}^j]_t = [x_{n+1/2}^j]_t + 2\gamma_n \vartheta \omega_{j,t}^{-1} ([\overline{x}_n]_t - [x_{n+1/2}^j]_t)$

      end

    end

  end

end

As we have seen, each computing unit $c \in \{1, \dots, C\}$ handles κ_c terms corresponding to the nodes in $\mathbb{V}_c$ of the hypergraph, and processes the functions $(g_j)_{j \in \mathbb{V}_c}$ associated with these nodes. Furthermore, a global synchronization step needs is required. This task could be assigned to one of the computing unit, e.g. the first one, as modelled in Figure 4 (bottom) by adding a fictitious term corresponding to hyperedge $\mathbb{V}_{C+1}$. This would however lead to a centralized scheme where the computing load between the different units would end up unbalanced.

A better strategy consists of distributing the operations performed on line 35 of Algorithm 4 over the different computing units. For this purpose, we first note that at iteration n, the c-th computing unit only needs the block components $([\overline{x}_n]_t)_{t \in \mathbb{T}_{\mathbb{V}_c}}$. In addition, because of Assumption 4.1-(ii)(a), some of these variables may be shared with the computing units $c-1$ (if $c \neq 1$) and $c+1$ (if $c \neq C$), where part of the variables $[x^j_{n+1/2}]_t$ necessary to compute the averages are locally available. As a consequence of Assumption 4.1-(ii)(b), no other variables than those available in either $\mathbb{T}_{\mathbb{V}_{c-1}} \cap \mathbb{T}_{\mathbb{V}_c}$ or $\mathbb{T}_{\mathbb{V}_c} \cap \mathbb{T}_{\mathbb{V}_{c+1}}$ are necessary . For example, if $c \neq 1$ and $t \in \mathbb{T}_{\mathbb{V}_{c-1}} \cap \mathbb{T}_{\mathbb{V}_c}$, the averaging operation reads

$$
\begin{aligned}
[\overline{x}_n]_t &= \frac{1}{|\mathbb{T}^*_t|} \sum_{j \in \mathbb{T}^*_t} [x^j_{n+1/2}]_t \\
&= \frac{1}{|\mathbb{T}^*_t|} \big([s_{n,c-1}]_t + [s_{n,c}]_t \big), \qquad (38)
\end{aligned}
$$

where

$$
[s_{n,c-1}]_t = \sum_{j \in \mathbb{V}_{c-1} \cap \mathbb{T}^*_t} [x^j_{n+1/2}]_t, \qquad (39)
$$

and $[s_{n,c}]_t$ is similarly defined. Since the variables $([x^j_{n+1/2}]_t)_{j \in \mathbb{V}_{c-1} \cap \mathbb{T}^*_t}$ are not available at unit c, the latter summation must be performed by unit $c-1$ and the result must be transmitted to unit c. This one will then be able to compute $[\overline{x}_n]_t$, so as to update variables $([x^j_{n+1}]_t)_{j \in \mathbb{V}_c \cap \mathbb{T}^*_t}$. Besides, $[\overline{x}_n]_t$ will be sent to unit $c-1$, which in turn will update its variables $([x^j_{n+1}]_t)_{j \in \mathbb{V}_{c-1} \cap \mathbb{T}^*_t}$. A similar

synchronization process can be followed for blocks with indices $t \in \mathbb{T}_{\mathbb{V}_c} \cap \mathbb{T}_{\mathbb{V}_{c+1}}$ with $c \neq C$. Finally, for the block indices t in $\mathbb{T}_{\mathbb{V}_c}$ which do not belong to $\mathbb{T}_{\mathbb{V}_{c-1}}$ or $\mathbb{T}_{\mathbb{V}_{c+1}}$,

$$[\overline{x}_n]_t = \text{mean}\left(([x^j_{n+1/2}]_t)_{j\in\mathbb{V}_c\cap\mathbb{T}^*_t}\right) = \frac{[s_{n,c}]_t}{|\mathbb{T}^*_t|}, \tag{40}$$

as we have then $|\mathbb{V}_c \cap \mathbb{T}^*_t| = |\mathbb{T}^*_t|$. This means that local averaging is only required for these blocks. In Figure 5, the synchronization workflow is summarized, while, in Algorithm 5, a more detailed account of the whole process is given.

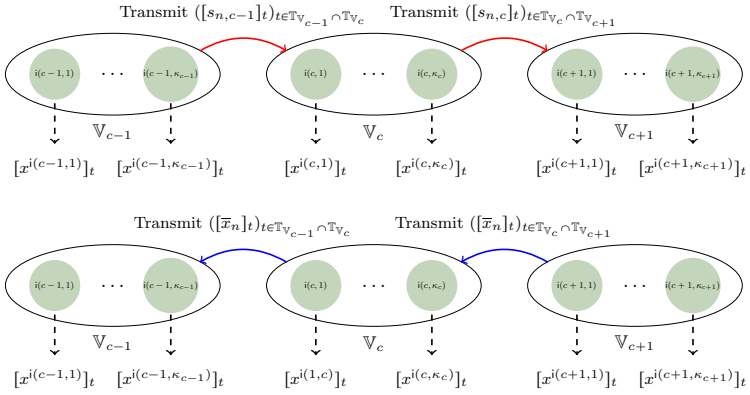


Figure 5: Global synchronisation process: Transmission of local summations to the next computing unit (top) ; Transmission of averaged blocks to the previous computing unit (bottom).

Remark 4.2.

(i) *It must be emphasized that, in order to facilitate the derivation of our algorithm, a common iteration variable n has been used for each computing unit. However, units have the flexibility to process data at their own speed. In particular, each unit may perform a different number of local synchronizations before a global one is made. In this sense, our algorithm is asynchronous. To understand why such behavior is allowed, it suffices to note that if no global synchronization arises and $\mathbb{J}_{n,c} = \varnothing$, then $(x^j_{n+1})_{j\in\mathbb{V}_c} = (x^j_n)_{j\in\mathbb{V}_c}$. This means that such a null iteration can be used to model a time when the c-th computing unit is idle while others are*

locally updating their variables.

(ii) *When the c-th computing unit operates a global synchronization, it will suspend its activities until it receives data from units* $c-1$ *(line 35) and/or* $c+1$ *(line 39), which happens only when these units also are globally synchronizing their variables. To ensure low latencies, global synchronization steps however have to be scheduled (quasi-)periodically for each computing unit based on their processing speeds (faster ones should schedule less frequent synchronizations than slower ones). Alternatively, when one unit decides to perform a global synchronization, it can broadcast a message to the others to warn them to do the same.*

(iii) *Other forms of local consensus could potentially be devised. For example, another choice would consist in setting* $L = 2C-1$ *and* $(\forall c \in \{1,\ldots,C-1\})$ $\mathbb{V}_{C+c} = \mathbb{V}_c \cup \mathbb{V}_{c+1}$. *Then, each node* $c \in \{1,\ldots,C-1\}$ *could be responsible for driving the synchronization with its neighbor of index* $c+1$. *However, it appears more difficult, in this context, to devise an efficient procedure to avoid deadlocks, contrary to our previous example.*

4.3. Application to video denoising

4.3.1. Observation model

In this section, we illustrate the performance of the proposed distributed algorithm for denoising video sequences. The original sequence $\overline{x} = ([\overline{x}]_t)_{1\leqslant t\leqslant T} \in \mathbb{R}^N$ is naturally decomposed in T blocks of data, each corresponding to one image composed of L pixels. The degradation model relating the observed noisy sequence $y = ([y]_t)_{1\leqslant t\leqslant T} \in \mathbb{R}^N$ to the sought sequence $\overline{x}$ with $TL = N$ is given by

$$(\forall t \in \{1,\ldots,T\}) \quad [y]_t = [\overline{x}]_t + [w]_t, \tag{41}$$

where $([w]_t)_{1\leqslant t\leqslant T} \in \mathbb{R}^N$ represents an additive zero-mean white Gaussian noise. An estimate of the unknown video can be inferred by solving Problem (5) where

Algorithm 5: Special case of distributed PDFB for the c-th computing unit

Setting of global constants:
$\mathbb{T}_j \equiv$ index set of blocks used at node $j \in \{1,\dots,J\}$
$\mathbb{T}_t^* \equiv$ index set of nodes using block $t \in \{1,\dots,T\}$
$\{\omega_{j,t} \mid 1 \leqslant j \leqslant J, t \in \mathbb{T}_j\} \subset (0,1]$ such that $(\forall t \in \{1,\dots,T\}) \ \sum_{j \in \mathbb{T}_t^*} \omega_{j,t} = 1$
$\vartheta = \min_{1 \leqslant j \leqslant J, 1 \leqslant t \leqslant T} \omega_{j,t}$, $\epsilon \in (0,1]$, $(\gamma_n)_{n\in\mathbb{N}}$ sequence of $[\epsilon, 2-\epsilon]$ with $\epsilon \in (0,1]$
Initialization:
$\mathbb{V}_c \equiv$ index set of nodes associated with computing unit c
$\mathbb{T}_{\mathbb{V}_c} \equiv$ set of block indices used in $\mathbb{V}_c$ (with the convention $\mathbb{T}_{\mathbb{V}_0} = \mathbb{T}_{\mathbb{V}_{C+1}} = \varnothing$)
$B_j \in \mathbb{R}^{M_j \times M_j}$ with $B_j \succeq \sum_{t \in \mathbb{T}_j} \omega_{j,t}^{-1} \mathcal{A}_{j,t} \mathcal{A}_{j,t}^\top, \quad j \in \mathbb{V}_c$
$\vartheta_c = \min_{j \in \mathbb{V}_c, t \in \mathbb{T}_j} \omega_{j,t}, \quad \ell \in \{1,\dots,C\}$
$y_0^j \in \mathbb{R}^{M_j}, [x_0^j]_t = [\widetilde{x}]_t - \omega_{j,t}^{-1} \mathcal{A}_{j,t}^\top y_0^j, \quad j \in \mathbb{V}_c, t \in \mathbb{T}_j.$
Main loop:
for $n = 0, 1, \dots$ **do**
  $\mathbb{J}_{n,c} \subset \mathbb{V}_c$
  for $j \in \mathbb{J}_{n,c}$ **do**
    $\widetilde{y}_n^j = y_n^j + \gamma_n B_j^{-1} \sum_{t \in \mathbb{T}_j} \mathcal{A}_{j,t} [x_n^j]_t$
    $y_{n+1}^j = \widetilde{y}_n^j - \gamma_n B_j^{-1} \mathrm{prox}_{\gamma_n B_j^{-1}, g_j}\big(\gamma_n^{-1} B_j \widetilde{y}_n^j\big)$
    for $t \in \mathbb{T}_j$ **do** $[x_{n+1/2}^j]_t = [x_n^j]_t - \omega_{j,t}^{-1} \mathcal{A}_{j,t}^\top (y_{n+1}^j - y_n^j)$;
  end
  for $j \in \mathbb{V}_c \backslash \mathbb{J}_{n,c}$ **do**
    $y_{n+1}^j = y_n^j$
    $([x_{n+1/2}^j]_t)_{t \in \mathbb{T}_j} = ([x_n^j]_t)_{t \in \mathbb{T}_j}$
  end
  for $t \in \mathbb{T}_{\mathbb{V}_c}$ **do** $[s_{n,c}]_t = \sum_{j \in \mathbb{V}_c \cap \mathbb{T}_t^*} [x_{n+1/2}^j]_t$;
  if *synchronization is local* **then**
    for $j \in \mathbb{V}_c$ **do**
      for $t \in \mathbb{T}_j$ **do**
        $[x_{n+1}^j]_t = [x_{n+1/2}^j]_t + \gamma_n \vartheta_c \, \omega_{j,t}^{-1} \Big(\frac{[s_{n,c}]_t}{|\mathbb{V}_c \cap \mathbb{T}_t^*|} - [x_{n+1/2}^j]_t \Big)$
      end
    end
  else
    Global synchronization:
    if $c \neq C$ **then send** $([s_{n,c}]_t)_{t \in \mathbb{T}_{\mathbb{V}_c} \cap \mathbb{T}_{\mathbb{V}_{c+1}}}$ to unit $c+1$;
    if $c \neq 1$ **then**
      wait for receiving $([s_{n,c-1}]_t)_{t \in \mathbb{T}_{\mathbb{V}_{c-1}} \cap \mathbb{T}_{\mathbb{V}_c}}$ from unit $c-1$
      for $t \in \mathbb{T}_{\mathbb{V}_{c-1}} \cap \mathbb{T}_{\mathbb{V}_c}$ **do** $[\overline{x}_n]_t = \frac{1}{|\mathbb{T}_t^*|} \big([s_{n,c-1}]_t + [s_{n,c}]_t \big)$;
      send $([\overline{x}_n]_t)_{t \in \mathbb{T}_{\mathbb{V}_{c-1}} \cap \mathbb{T}_{\mathbb{V}_c}}$ to unit $c-1$
    end
    if $c \neq C$ **then wait for receiving** $([\overline{x}_n]_t)_{t \in \mathbb{T}_{\mathbb{V}_c} \cap \mathbb{T}_{\mathbb{V}_{c+1}}}$ from unit $c+1$;
    for $t \in \mathbb{T}_{\mathbb{V}_c} \backslash (\mathbb{T}_{\mathbb{V}_{c-1}} \cup \mathbb{T}_{\mathbb{V}_{c+1}})$ **do** $[\overline{x}_n]_t = \frac{[s_{n,c}]_t}{|\mathbb{T}_t^*|}$;
    for $j \in \mathbb{V}_c$ **do**
      for $t \in \mathbb{T}_j$ **do**
        $[x_{n+1}^j]_t = [x_{n+1/2}^j]_t + \gamma_n \vartheta \omega_{j,t}^{-1} \big([\overline{x}_n]_t - [x_{n+1/2}^j]_t \big)$
      end
    end
  end
end

$J = T$ and $\widetilde{x} = y$. Note that similar applicative problems have been addressed by using other techniques in [39]. The last quadratic term in (5) is a least squares data fidelity term ensuring compliance with model (41), and functions $(g_j)_{1 \leqslant j \leqslant T}$ stand for regularization functions that incorporate both temporal and spatial prior knowledge on each video frame. The temporal regularization is fulfilled by taking into account motion compensation between the previous and next neighbouring frames. More precisely, at each time $t \in \{2, \dots, T-1\}$, the linear operator A_t extracts the current frame x_t and its neighbors (x_{t-1}, x_{t+1}) as shown by:

$$[[x]_1 \dots [x]_{t-1}[x]_t[x]_{t+1} \dots [x]_T] \overset{A_t}{\to} [[x]_{t-1}[x]_t[x]_{t+1}]\,. \tag{42}$$

The linear operators $(A_t)_{1 \leqslant t \leqslant T}$ thus have the block sparse structure expressed by (32) with

$$(\forall t \in \{1, \dots, T\}) \quad \mathbb{T}_t = \big\{ \max\{t-1, 1\}, t, \min\{t+1, T\} \big\} \tag{43}$$

and

$$\mathcal{A}_{1,1} = \begin{bmatrix} I_L & 0 \end{bmatrix}^\top, \quad \mathcal{A}_{1,2} = \begin{bmatrix} 0 & I_L \end{bmatrix}^\top, \tag{44}$$

$$(\forall t \in \{2, \dots, T-1\}) \quad \mathcal{A}_{t,t-1} = \begin{bmatrix} I_L & 0 & 0 \end{bmatrix}^\top \tag{45}$$

$$\mathcal{A}_{t,t} = \begin{bmatrix} 0 & I_L & 0 \end{bmatrix}^\top \tag{46}$$

$$\mathcal{A}_{t,t+1} = \begin{bmatrix} 0 & 0 & I_L \end{bmatrix}^\top \tag{47}$$

$$\mathcal{A}_{T,T-1} = \begin{bmatrix} I_L & 0 \end{bmatrix}^\top, \quad \mathcal{A}_{T,T} = \begin{bmatrix} 0 & I_L \end{bmatrix}^\top. \tag{48}$$

For every $t \in \{1, \dots T\}$, each regularization function $g_t \colon \mathbb{R}^{M_t} \to [0, +\infty)$ is

convex, proper, lower semi-continuous and such that

$$M_t = \begin{cases} 3L & \text{if } t \neq 1 \text{ and } t \neq T \\ 2L & \text{otherwise,} \end{cases} \tag{49}$$

and, for every $x = ([x]_t)_{1 \leqslant t \leqslant T}$,

$$g_t\left(([x]_{t'})_{t' \in \mathbb{T}_t}\right) \;=\; \eta \, \mathrm{tgv}([x]_t) \;+\; \iota_{[x_{\min}, x_{\max}]^L}([x]_t) \;+\; h_t\left(([x]_{t'})_{t' \in \mathbb{T}_t}\right), \tag{50}$$

where "tgv" denotes the *Total Generalized Variation* regularization from [40], defined as

$$(\forall z \in \mathbb{R}^L) \quad \mathrm{tgv}(z) = \min_{d \in \mathbb{R}^{2L}} \; \alpha_0 \, \chi_2\left(Dz - d\right) + \alpha_1 \, \chi_3\left(\mathsf{G} d\right), \tag{51}$$

with $(\alpha_0, \alpha_1) \in (0, +\infty)^2$, $D \in \mathbb{R}^{2L \times L}$ is the concatenation of the horizontal and vertical spatial gradient operators:

$$D = \begin{bmatrix} \nabla_{\mathsf{H}} \\ \nabla_{\mathsf{V}} \end{bmatrix}, \quad \text{with} \quad \nabla_{\mathsf{H}} \in \mathbb{R}^{L \times L}, \quad \nabla_{\mathsf{V}} \in \mathbb{R}^{L \times L}, \tag{52}$$

and $\mathsf{G} \in \mathbb{R}^{3L \times 2L}$ is the Jacobian operator given by

$$\mathsf{G} = \begin{bmatrix} \nabla_{\mathsf{H}} & \nabla_{\mathsf{V}} & 0 \\ 0 & \nabla_{\mathsf{H}} & \nabla_{\mathsf{V}} \end{bmatrix}^{\top}, \tag{53}$$

while, for every $q \in \mathbb{N}^*$, $\chi_q \colon \mathbb{R}^{qL} \to \mathbb{R}$ is given by

$$\left(\forall (z_1, \ldots, z_q) \in (\mathbb{R}^L)^q\right) \quad \chi_q(z_1, \ldots, z_q) = \sum_{k=1}^{L} \sqrt{(z_{1,k})^2 + \cdots + (z_{q,k})^2}. \tag{54}$$

The indicator function $\iota_{[x_{\min}, x_{\max}]^L}$ in (50) imposes a range $[x_{\min}, x_{\max}]$ on the pixel values in each frame. In addition, h_t is a function introducing a temporal

regularization of the form

$$h_t\left(([x]_{t'})_{t'\in\mathbb{T}_t}\right) = \begin{cases} \beta_{t-1,t}\,\chi_1([x]_t - \mathcal{M}_{t-1\to t}\,[x]_{t-1}) \\ \qquad + \beta_{t+1,t}\,\chi_1([x]_t - \mathcal{M}_{t+1\to t}\,[x]_{t+1}) \\ \qquad\qquad\qquad \text{if } t \neq 1 \text{ and } t \neq T \\ \beta_{2,1}\,\chi_1([x]_1 - \mathcal{M}_{2\to 1}\,[x]_2) \\ \qquad\qquad\qquad \text{if } t = 1 \\ \beta_{T-1,T}\,\chi_1([x]_T - \mathcal{M}_{T-1\to T}\,[x]_{T-1}) \\ \qquad\qquad\qquad \text{if } t = T, \end{cases} \tag{55}$$

where $\mathcal{M}_{t-1\to t} \in \mathbb{R}^{L\times L}$ (resp. $\mathcal{M}_{t+1\to t} \in \mathbb{R}^{L\times L}$) is a motion compensation operator between the reference frame x_{t-1} (resp. x_{t+1}) and the current frame x_t, defined as described in [25, Section 5.2.2] (See also [41]). Finally, η, $(\beta_{t-1,t})_{2\leqslant t\leqslant T}$ and $(\beta_{t+1,t})_{1\leqslant t\leqslant T-1}$ are positive regularization parameters controlling the strength of the contribution of their associated terms. The values of these parameters were optimized by grid search so as to achieve the best denoising performance.

4.3.2. Proposed method

We employ our proposed asynchronous distributed framework to address the previous denoising problem. More precisely, we use the practical implementation detailed in Algorithm 5. Functions $(g_t)_{1\leqslant t\leqslant T}$ and their associated primal variables $([x^t]_{t'})_{t'\in\mathbb{T}_t}$ for $t \in \{1,\ldots,T\}$, are spread over C computing units, each of them handling $\kappa_c \in \{1,\ldots,T\}$ nodes with $\sum_{c=1}^{C}\kappa_c = T$.The associated hyperedges are then given by

$$(\forall c \in \{1,\ldots,C\}) \qquad \mathbb{V}_c = \{(c-1)\kappa_c + 1,\ldots,c\kappa_c\}. \tag{56}$$

Note that, since

$$(\forall c \in \{1,\ldots,C\}) \qquad \mathbb{T}_{\mathbb{V}_c} = \big\{\max\{(c-1)\kappa_c,1\},\ldots,\min\{c\kappa_c+1,T\}\big\}, \tag{57}$$

we have

$$(\forall c \in \{1, \dots, C-1\}) \qquad \mathbb{T}_{\mathbb{V}_c} \cap \mathbb{T}_{\mathbb{V}_{c+1}} = \{c\kappa_c, c\kappa_c + 1\}, , \tag{58}$$

so that Assumption 4.1 holds provided that $\kappa_c > 1$.

In the local optimization first performed at the n-th iteration of Algorithm 5, we used, for every $j \in \{1, \dots, T\}$, $B_j = \sum_{t \in \mathbb{T}_j} \omega_{j,t}^{-1} I_{M_j}$ and $\gamma_n \equiv 1.7$. Then, the local or global synchronization steps are performed as described in Section 4.2. In our case, for every $t \in \{1, \dots, T\}$, $\mathbb{T}_t^* = \mathbb{T}_t$. If $t \in \mathbb{T}_{\mathbb{V}_c}$ with $c \in \{1, \dots, C\}$ corresponds neither to the smallest nor the largest index in $\mathbb{V}_c$, then 3 values need to be summed to compute $[s_{n,c}]_t$. If t is the smallest or the largest index in $\mathbb{V}_c$, then the summation involves only two terms. Finally, if $c > 1$ and $t = (c-1)\kappa_c$ (resp. $c < C$ and $t = c\kappa_c + 1$)) , then $[s_{n,c}]_t = [x_{n+1/2}^{t+1}]_t$ (resp. $[s_{n,c}]_t = [x_{n+1/2}^{t-1}]_t$). In global synchronization steps, by virtue of (58), only variables $[s_{n,c}]_{c\kappa_c}$ and $[s_{n,c}]_{c\kappa_c+1}$ are transmitted from computing unit $c \neq C$ to computing unit $c+1$, which in return sends back the updated averages $[\overline{x}_n]_{c\kappa_c}$ and $[\overline{x}_n]_{c\kappa_c+1}$. This workflow is illustrated in Figures 6 and 7 by an example showing two computing units both handling $\kappa = 3$ nodes.

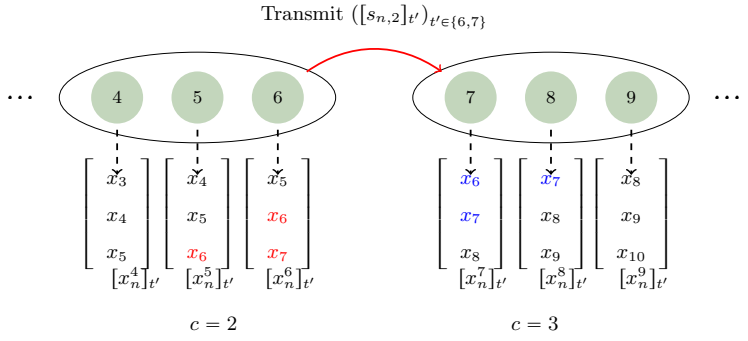


Figure 6: Transmission of local sums $([s_{n,2}]_{t'})_{t' \in \{6,7\}}$ shared between $\mathbb{T}_{\mathbb{V}_2} = \{3, 4, 5, 6, 7\}$ and $\mathbb{T}_{\mathbb{V}_3} = \{6, 7, 8, 9, 10\}$ from computing unit $c = 2$ to computing unit $c = 3$.

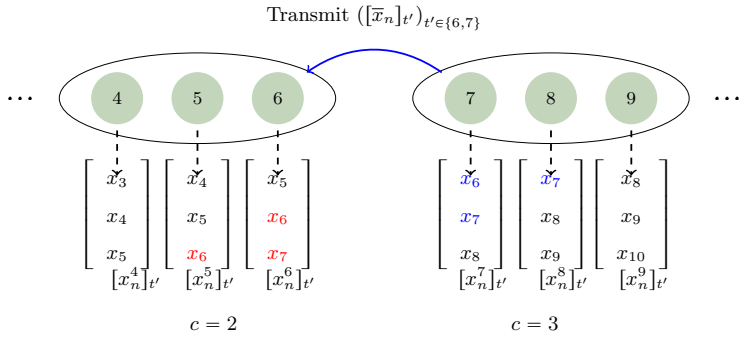


Figure 7: Transmission of averaged images $([\overline{x}_n]_{t'})_{t'\in\{6,7\}}$ from computing unit $c = 3$ to computing unit $c = 2$.

In our simulations, the global synchronizations are activated every 4 iterations. This synchronization frequency was chosen in order to achieve a reasonable trade-off between the communication overhead and a satisfactory convergence speed. The weights $(\omega_{j,t})_{1\leqslant t\leqslant T, j\in\mathbb{T}_t^*}$ are set to $\frac{1}{|\mathbb{T}_t^*|}$.

4.3.3. Simulation results

The performance of the proposed denoising method are evaluated on four standard test video sequences from `http://trace.eas.asu.edu/yuv/`, namely **Flower**, **Bus**, **Stefan**, and **Irene** with $T = 72$ frames. These are YCbCr-colored frames of size 352×288. The four videos display significantly high motions which makes their restoration challenging. For the sake of fair comparison, we applied the restoration methods (ours and BM4D) only on the luminance channel. All the displayed SNR (signal-to-noise ratio) scores are computed on luminance channel only. The restored color videos, displayed at the end of this section, are obtained by further restoring the chrominance channels with a median filter of size 3×3. The degraded videos are obtained by adding zero-mean white Gaussian noise to the original video sequences, resulting in an initial SNR (on luminance channel) of 25.02 dB, 24.84 dB, 24.41 dB and 25.51 dB for the four sequences respectively. Parameters α_0 and α_1 in (51) were set to $\alpha_0 = 0.7108$ and $\alpha_1 = 1 - \alpha_0$. Moreover, the weights of the spatial and temporal regularizations in (50) have been set constant through the images, for each video. They

have been optimized by grid search of about 100 values, so as to maximize the restored video SNR. The selected values are listed in Table 1.

Sequences	Spatial regularization	Temporal regularization
Flower	$\eta = 2.9 \times 10^{-2}$	$\begin{cases} (\beta_{t-1,t})_{2 \leqslant t \leqslant T-1} = 6.01 \times 10^{-2} \\ \beta_{2,1} = \beta_{T-1,T} = 9.13 \times 10^{-1} \end{cases}$
Bus	$\eta = 2.02 \times 10^{-2}$	$\begin{cases} (\beta_{t-1,t})_{2 \leqslant t \leqslant T-1} = 2.81 \times 10^{-2} \\ \beta_{2,1} = \beta_{T-1,T} = 9.54 \times 10^{-2} \end{cases}$
Stefan	$\eta = 1.47 \times 10^{-3}$	$\begin{cases} (\beta_{t-1,t})_{2 \leqslant t \leqslant T-1} = 6.62 \times 10^{-2} \\ \beta_{2,1} = \beta_{T-1,T} = 9.6 \times 10^{-1} \end{cases}$
Irene	$\eta = 2.04 \times 10^{-2}$	$\begin{cases} (\beta_{t-1,t})_{2 \leqslant t \leqslant T-1} = 0.75 \times 10^{-2} \\ \beta_{2,1} = \beta_{T-1,T} = 1.5 \times 10^{-2} \end{cases}$

Table 1: Values of the spatial and temporal regularization parameters.

Our method is implemented with Julia-1.4.1 and a *Message Passing Interface* (MPI) wrapper for managing communication between cores [42, 43]. The Julia `MAT` and `HDF5` packages (version 1.10.6) are used to read/write input/output data. For parallel processing, we use the Intel MPI v2019.3.199 compiler paired with the Julia `MPI` package. Receive (`recv`) and send (`isend`) operations are used to communicate between different cores when running our algorithm, and as a last step to gather the restored frames into a full restored video. For ease of reproducibility, our code is made available at `https://github.com/MarinENSTA/distributed_julia_denoising`. We use a multicore architecture with 170 Gb RAM, using 2 Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz, each with 20 cores and 2 threads per core, hence C can be up to 40 cores. The experiments are run using 30 iterations of Algorithm 5, which is sufficient to reach convergence. We evaluate the proposed distributed approach in terms of restoration quality and acceleration provided by our algorithm with respect to the number of computing units. We consider various

values of $C \in \{1, \ldots, 36\}$. If $T = \kappa C$ for $\kappa \in \mathbb{N}^*$, then the images composing the video sequences are partitioned in groups of equal size κ. Otherwise, we decompose the sequence in a balanced way. For instance, let $C = 19$. Then cores $c \in \{1, \ldots, 4\}$ process blocks of 3 images while cores $c \in \{5, \ldots, 19\}$ handle blocks of 4 images. All the provided computation times are obtained using the `$SECONDS` Linux environment variable. This is an independent time observer meant to provide the time cost of the complete video denoising process, regardless of parallel vs sequential parts of the algorithm. Note that the motion compensation operators involved in our regularization function are pre-computed prior to running our algorithm, by applying the optical flow estimation C++ software from [44] on the noisy sequence. This step is not included in our computational time evaluation, for a fair and comprehensive scalability analysis.

Our method achieves satisfactory restoration results with an improvement of 3.25 dB for **Flower**, 2.97 dB for **Bus**, 4.27 dB for **Stefan** and 5.6 dB for **Irene**, with respect to the degraded video. Moreover, according to our observations, the convergence to the sought solution was reached in each experiment regardless the number of used cores. Otherwise stated, the quality of the solution is identical, whatever the number of cores activated. Figures 8, 9, 10, and 11 show some frames illustrative of the degraded and restored sequences (after including the filtered chrominance channels). These demonstrate the good visual quality of the performed denoising, even in some examples in the presence of large motion.

Figure 12 shows the speedup in execution time with respect to the number of cores, which is estimated as follows:

$$\text{Speedup for } C \text{ cores} \quad = \quad \frac{\text{Execution time with 1 core}}{\text{Execution time with } C \text{ cores}}. \tag{59}$$

Figure 12 shows how speedup increases as we increase the number of cores from $C = 1$ to $C = 36$. The behavior is similar for all video tested. As the number of cores increases, the speedup gets farther to the ideal (linear) curve. This is a consequence of the number of cores increasing, which results

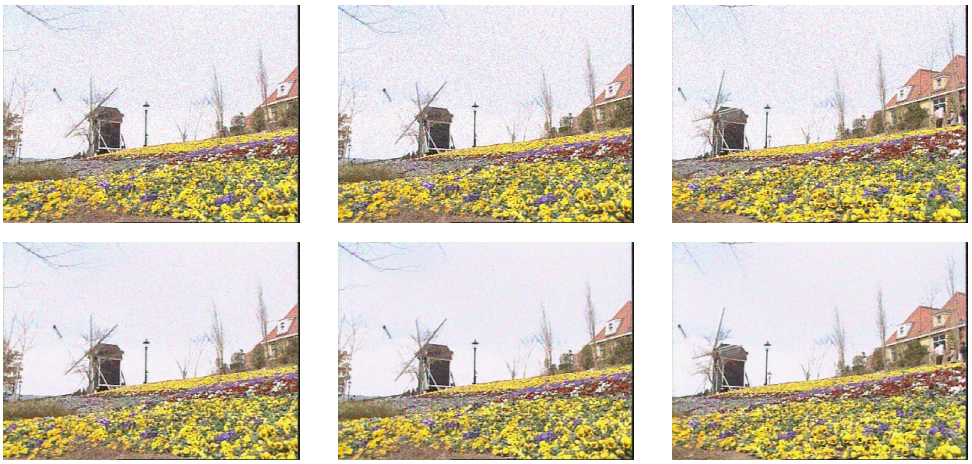


Figure 8: **Flower** sequence: Input degraded images (top) initial SNR = 25.02 dB, associated restored images (bottom) final SNR = 28.27 dB.

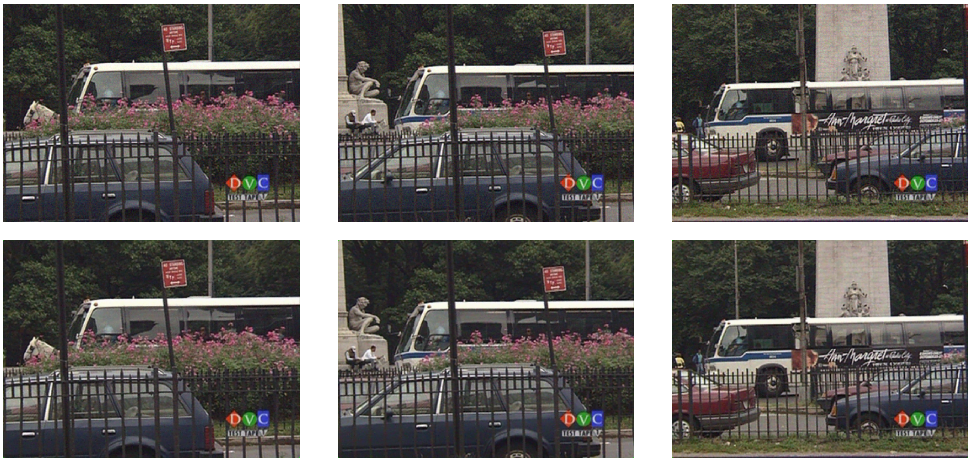


Figure 9: **Bus** sequence: Input degraded images (top) initial SNR = 24.84 dB, associated restored images (bottom) final SNR = 27.81 dB.

in higher communication time costs between parallel processes thus impacting global execution time and impeding better time improvements for the algorithm. One can also notice a non-monotonic behavior in the speedup curves. This is explained by a slight decrease of the overall performance when the number of video frames is not a multiple of the core numbers, thus yielding an imperfect balance of the workload per core. We furthermore display in Figure 13 the

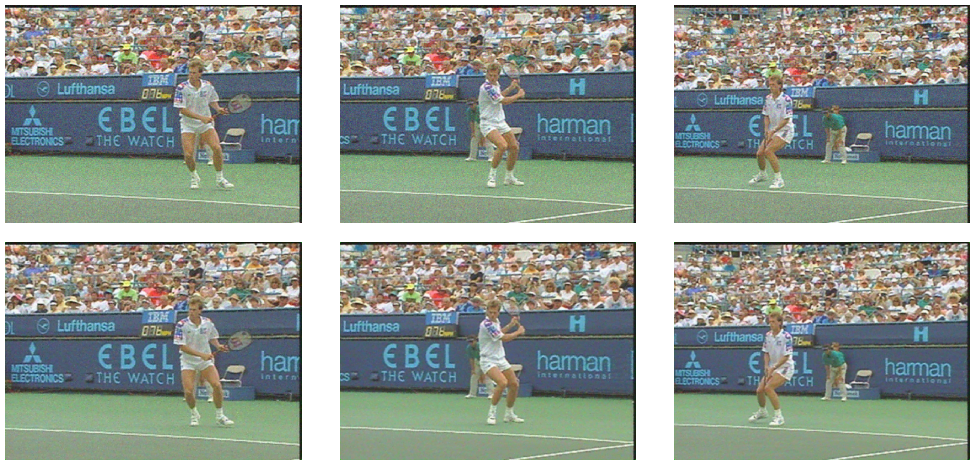
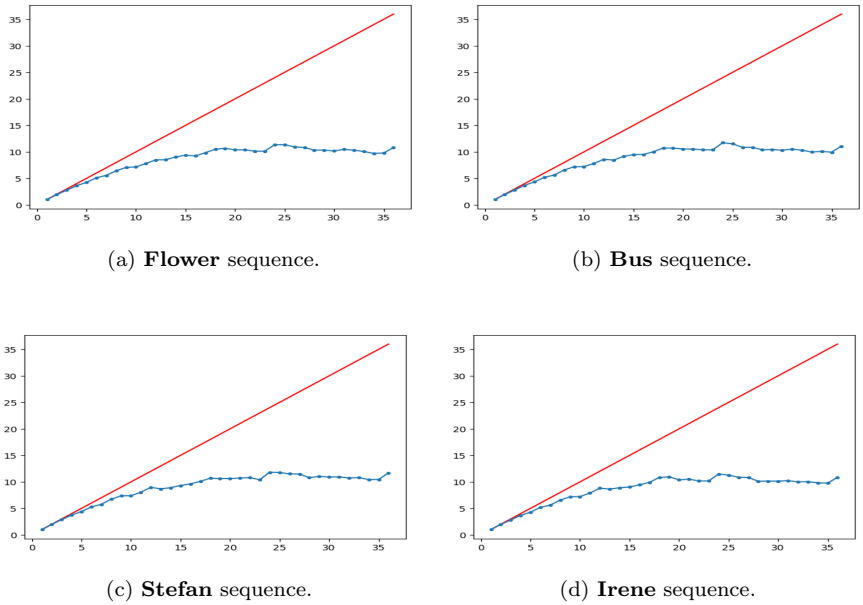


Figure 10: **Stefan** sequence: Input degraded images (top) initial SNR = 24.41 dB, associated restored images (bottom) final SNR = 28.68 dB.

Figure 11: **Irene** sequence: Input degraded images (top) initial SNR = 25.51 dB, associated restored images (bottom) final SNR = 31.11 dB.

averaged computational time per frame, with respect to the number of cores. For instance, the restoration of Flower sequence requires from 43.1 seconds per frame (for one core) to 4.2 seconds per frame (for 36 cores).

We additionally performed a comparison with the method BM4D [39] also aiming at restoring video sequences. BM4D relies on a totally different approach, based on non-local filtering. The obtained results in terms of averaged SNR per

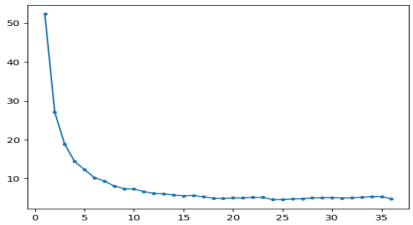


(a) **Flower** sequence. (b) **Bus** sequence.

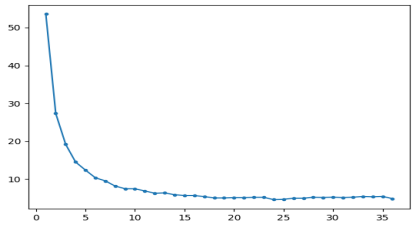
(c) **Stefan** sequence. (d) **Irene** sequence.

Figure 12: Speedup (y-axis) with respect to the number of used cores (x-axis): proposed method (blue), linear speedup (red).

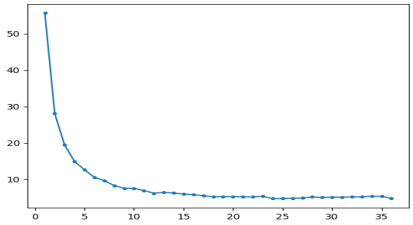
frame are shown in Table 2. We can observe that our method achieves competitive, and sometimes superior, performance in terms of quality metrics. In terms of computational time, when run on Flower video using the same computer, BM4D requires 6.3 seconds per frame, which is slightly slower than the best performance reached by our method. For BM4D, we relied on the publicly available code `https://webpages.tuni.fi/foi/GCF-BM3D/BM4D_v3p2.zip`, combining Matlab (using th R2020b version) and precompiled C codes in mex format. Such implementation of BM4D does not exploit multi-CPU, and the same computing time was observed whatever the number of active cores. A fair time complexity comparison would however require BM4D to be developed using the same programming language, in addition to a rigorous management of implicit and explicit multithreading aspects.



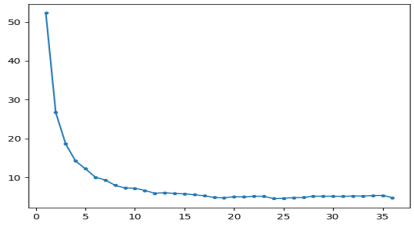
(a) **Flower** sequence.



(b) **Bus** sequence.



(c) **Stefan** sequence.



(d) **Irene** sequence.

Figure 13: Averaged computing time per frame in seconds (y-axis) with respect to the number of used cores (x-axis).

Table 2: Comparison in terms of averaged SNR per frame.

Sequences	Initial SNR	Our method	BM4D
Flower	25.02 dB	28.27 dB	27.87 dB
Bus	24.84 dB	27.81 dB	25.65 dB
Stefan	24.41 dB	28.68 dB	28.37 dB
Irene	25.51 dB	31.11 dB	30.86 dB

5. Conclusion

This article has introduced a fully parallelized version of the preconditioned dual block-coordinate forward-backward algorithm for computing proximity operators. Our algorithm benefits from all the advantages of primal-dual methods and the acceleration provided by a block-coordinate strategy combined with a variable metric approach. We mainly focused on an instance of the proposed approach for which we proposed a practical asynchronous implementation, as-

suming that a given number of computing units is available. Although our distributed algorithm can be applied to a wide range of problems, we investigated its application to video sequence denoising. The experimental results we obtained are quite promising and demonstrate the ability of our algorithm to take advantage of multiple cores. An acceleration of about 12 was reached with a standard two-processors computer configuration. In future works, we intend to experiment different distributed implementations based on other partitioning strategies and hypergraph topologies and to study the application of our distributed framework to other proximal optimization algorithms.

Acknowledgments

This work was supported by the Institut Universitaire de France, and by the European Research Council Starting Grant MAJORIS ERC-2019-STG-850925.

References

[1] P. L. Combettes, J.-C. Pesquet, Proximal splitting methods in signal processing, in: H. H. Bauschke, R. Burachik, P. L. Combettes, V. Elser, D. R. Luke, H. Wolkowicz (Eds.), Fixed-Point Algorithms for Inverse Problems in Science and Engineering, Springer-Verlag, New York, 2010, pp. 185–212.

[2] N. Parikh, S. Boyd, Proximal algorithms, Found. Trends Optim. 1 (3) (2014) 127–239.

[3] N. Komodakis, J.-C. Pesquet, Playing with duality : An overview of recent primal-dual approaches for solving large-scale optimization problems, IEEE Signal Process. Mag. 32 (6) (2015) 31–54.

[4] P. L. Combettes, D. Dung, B. C. Vũ, Dualization of signal recovery problems, Set-Valued Var. Anal. 18 (3) (2010) 373–404.

[5] L. Condat, A primal-dual splitting method for convex optimization involving lipschitzian, proximable and linear composite terms, J. Optim. Theory App. 158 (2) (2013) 460–479.

[6] S. R. Becker, P. L. Combettes, An algorithm for splitting parallel sums of linearly composed monotone operators, with applications to signal recovery, J. Nonlinear Convex Anal. 15 (1) (2014) 137–159.

[7] C. Couprie, L. Grady, L. Najman, J.-C. Pesquet, H. Talbot, Dual constrained tv-based regularization on graphs, SIAM J. Imaging Sci. 6 (2013) 1246–1273.

[8] A. Jezierska, E. Chouzenoux, J.-C. Pesquet, H. Talbot, A primal-dual proximal splitting approach for restoring data corrupted with poisson-gaussian noise, in: IEEE Int. Conf. Acoust. Speech and Signal Process. (ICASSP 2012), Kyoto, Japan, 2012, pp. 1085–1088.

[9] A. Onose, R. E. Carrillo, A. Repetti, J. D. McEwen, J.-T. Thiran, J.-C. Pesquet, Y. Wiaux, Scalable splitting algorithms for big-data interferometric imaging in the SKA era, Monthly Notices of the Royal Astronomical Society 462 (4) (2016) 4314–4335.

[10] A. Chambolle, T. Pock, A first-order primal-dual algorithm for convex problems with applications to imaging, J. Math. Imag. Vision 40 (1) (2010) 120–145.

[11] P. L. Combettes, L. Condat, J.-C. Pesquet, B. C. Vũ, A forward-backward view of some primal-dual optimization methods in image recovery, in: IEEE Int. Conf. Image Process. (ICIP 2014), Paris, France, 2014, pp. 4141–4145.

[12] R. I. Boţ, C. Hendrich, Convergence analysis for a primal-dual monotone + skew splitting algorithm with applications to total variation minimization, J. Math. Imaging Vision 49 (3) (2014) 551–568.

[13] P. L. Combettes, J.-C. Pesquet, Primal-dual splitting algorithm for solving inclusions with mixtures of composite, Lipschitzian, and parallel-sum type monotone operators, Set-Valued Var. Anal. 20 (2) (2012) 307–330.

[14] R. I. Boţ, C. Hendrich, A Douglas-Rachford type primal-dual method for solving inclusions with mixtures of composite and parallel-sum type monotone operators, SIAM J. Optim. 23 (4) (2013) 2541–2565.

[15] P. L. Combettes, J.-C. Pesquet, Stochastic quasi-Fejér block-coordinate fixed point iterations with random sweeping, SIAM J. Optim. 25 (2) (2015) 1221–1248.

[16] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Found. Trends Machine Learn. 8 (1) (2011) 1–122.

[17] L. M. Briceño Arias, P. L. Combettes, A monotone+skew splitting model for composite monotone inclusions in duality, SIAM J. Optim. 21 (4) (2011) 1230–1250.

[18] F. Iutzeler, P. Bianchi, P. Ciblat, W. Hachem, Explicit convergence rate of a distributed alternating direction method of multipliers, IEEE Trans. Autom. Control 61 (4) (2016) 892–904.

[19] D. Davis, Convergence rate analysis of primal-dual splitting schemes, SIAM J. Optim. 25 (3) (2015) 1912–1943.

[20] L. Rosasco, S. Villa, B. C. Vũ, A first-order stochastic primal-dual algorithm with correction step, Numer. Funct. Anal. Optim. 38 (5) (2017) 602–626.

[21] S. Shalev-Shwartz, T. Zhang, Stochastic dual coordinate ascent methods for regularized loss minimization, J. Mach. Learn. Res. 14 (1) (2013) 567–599.

[22] P. Bianchi, W. Hachem, F. Iutzeler, A coordinate descent primal-dual algorithm and application to distributed asynchronous optimization, IEEE Trans. Autom. Control 61 (10) (2016) 2947–2957.

[23] Z. Qu, P. Richtárik, T. Zhang, Randomized dual coordinate ascent with arbitrary sampling, in: Adv. Neural Inf. Process. Syst. (NIPS 2015), Montréal, Canada, 2015, pp. 865–873.

[24] M. Jaggi, V. Smith, M. Takac, J. Terhorst, S. Krishnan, T. Hofmann, M. I. Jordan, Communication-efficient distributed dual coordinate ascent, in: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K. Weinberger (Eds.), Adv. Neural Inf. Process. Syst., Curran Associates, Inc., 2014, pp. 3068–3076.

[25] F. Abboud, E. Chouzenoux, J.-C. Pesquet, J.-H. Chenot, L. Laborelli, Dual block coordinate forward-backward algorithm with application to deconvolution and deinterlacing of video sequences, J. Math. Imaging Vision 59 (3) (2017) 415–431.

[26] A. Chambolle, T. Pock, A remark on accelerated block coordinate descent for computing the proximity operators of a sum of convex functions, SIAM J. Comput. Math. 1 (2015) 29–57.

[27] J.-C. Pesquet, A. Repetti, A class of randomized primal-dual algorithms for distributed optimization, J. Nonlinear Convex Anal. 16 (12) (2015) 2453–2490.

[28] P. Richtárik, M. Takác, Distributed coordinate descent method for learning with big data, J. Mach. Learn. Res. 17 (75) (2016) 1–25.

[29] J. Xu, Y. Sun, Y. Tian, G. Scutari, A unified contraction analysis of a class of distributed algorithms for composite optimization, https://arxiv.org/abs/1910.09817 (2019).

[30] E. Chouzenoux, J.-C. Pesquet, A. Repetti, Variable metric forward-backward algorithm for minimizing the sum of a differentiable function and a convex function, J. Optim. Theory App. 162 (1) (2014) 107–132.

[31] E. Chouzenoux, J.-C. Pesquet, A. Repetti, A block coordinate variable metric forward-backward algorithm, J. Global Optim. 66 (3) (2016) 457–485.

[32] S. Becker, J. Fadili, A quasi-Newton proximal splitting method, in: Adv. Neural Inf. Process. Syst. (NIPS 2012), Lake Tahoe, Nevada, 2012, pp. 2627–2635.

[33] J. J. Moreau, Proximité et dualité dans un espace hilbertien, Bull. Soc. Math. France 93 (1965) 273–299.

[34] P. L. Combettes, B. C. Vũ, Variable metric forward-backward splitting with applications to monotone inclusions in duality, Optimization 63 (9) (2014) 1289–1318.

[35] P. L. Combettes, D. Dũng, B. C. Vũ, Proximity for sums of composite functions, J. Math. Anal. Appl. 380 (2) (2011) 680–688.

[36] A. Chambolle, P. Tan, S. Vaiter, Accelerated alternating descent methods for dykstra-like problems, Journal of Mathematical Imaging and Vision 59 (3) (2017) 481–497.

[37] N. Pustelnik, C. Chaux, J.-C. Pesquet, Parallel proximal algorithm for image restoration using hybrid regularization, IEEE Trans. Image Process. 20 (9) (2011) 2450–2462.

[38] A. Nedic, A. Ozdaglar, P. A. Parrilo, Constrained consensus and optimization in multi-agent networks, IEEE Trans. Autom. Control 55 (4) (2010) 922–938.

[39] A. Maggioni, G. Boracchi, A. Foi, K. Egiazarian, Video denoising, deblocking, and enhancement through separable 4-d nonlocal spatiotemporal transforms, IEEE Trans. Image Process. 21 (9) (2012) 3952–3966.

[40] K. Bredies, K. Kunisch, T. Pock, Total generalized variation, SIAM J. Imaging Sci. 3 (3) (2010) 492—-526.

[41] F. Abboud, E. Chouzenoux, J.-C. Pesquet, J.-C. Chenot, L. Laborelli, An alternating proximal approach for blind video deconvolution, Signal Processing: Image Communication 70 (2019) 21–36.

[42] M. P. I. Forum, MPI: A Message-Passing Interface standard, Tech. rep. (1994).

[43] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message Passing Interface, 2nd Edition, MIT Press, 1999.

[44] C. Liu, W. T. Freeman, E. H. Adelson, Y. Weiss, Human-assisted motion annotation, in: IEEE Conf. Comput. Vis. Pattern Recogn. (CVPR 2008), Anchorage, AK, 2008, pp. 1–8.