Distributed Algorithms for Proximity Operator Computation with Applications to Video Processing

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

Distributed Algorithms for Proximity Operator Computation with Applications to Video Processing

Feriel Abboud, *Member, IEEE,* Emilie Chouzenoux, *Member, IEEE,*
Jean-Christophe Pesquet, *Fellow, IEEE,* and Hugues Talbot, *Member, IEEE*

Abstract

In this paper, we present a new distributed algorithm for computing the proximity operator of a sum of non-necessarily differentiable convex functions composed with arbitrary linear operators. Each involved function is associated with a node of a hypergraph, with the ability to communicate with neighboring nodes sharing the same hyperedge. Our algorithm relies on a primal-dual splitting strategy with established convergence guaranties. We show how it can be efficiently implemented to take full advantage of a multi-core architecture. The good numerical performance of the proposed approach is illustrated with a problem of video sequence denoising, where a significant speedup is achieved.

Index Terms

Convex optimization, proximal methods, video restoration, parallel programming.

I. Introduction

Numerous problems in data science such as video restoration require the processing of huge 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/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 sum 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]–[6]. The main advantage of many of these algorithms is that none of the linear operators needs to be inverted which makes this class of algorithms well suited for large-scale problems

F. Abboud is with WITBE France (e-mail: feriel.abboud@gmail.com).

E. Chouzenoux is with University Paris-East LIGM, UMR CNRS 8049, and with GALEN team, INRIA Saclay, Center for Visual Computing, CentraleSupelec, Gif-sur-Yvette, France (e-mail: emilie.chouzenoux@u-pem.fr).

J.-C. Pesquet and H. Talbot are with Center for Visual Computing, CentraleSupelec, University Paris-Saclay, Gif-sur-Yvette, France (e-mail: jean-christophe@pesquet.eu, hugues.talbot@esiee.fr).

encountered in various application fields [7]–[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]–[20]. Moreover, primal-dual algorithms have been combined with a block-coordinate approach recently, 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 reasonable 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 refereed to as dual ascent methods.

The aforementioned algorithms were originally proposed 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], [18], [27], [28], 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.

In this paper, we propose a new distributed algorithm for computing the proximity operator of a sum of convex functions involving linear operators. The proposed algorithm extends the dual block preconditioned forward-backward algorithm that was recently proposed in [25] to a distributed asynchronous scenario. Each involved function is now considered as locally related to a node of a connected hypergraph, where communications are allowed between neighboring nodes that share the same hyperedge. Our method takes advantage of variable metric techniques that have been shown to be efficient for accelerating the convergence speed of proximal approaches [29]–[31]. Our proposal also benefits from the classical key advantages of primal-dual splitting strategies, in particular their ability to handle a finite sum of convex functions without inverting none of the involved linear operators, and its convergence is guaranteed.

The remainder of this paper is organized as follows: in Section II we recall some fundamental background and present a centralized dual block-coordinate forward-backward algorithm for computing proximity operators. In Section III, we introduce an asynchronous version of this algorithm. In Section IV, we discuss a useful special case and we describe its practical implementation on a distributed architecture. Section V shows the good performance of the proposed algorithm in the context of video denoising problems. Finally, some conclusions are drawn in Section VI.

II. Problem formulation

A. 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 [1], [32]:

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

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 the identity matrix, the standard proximity operator prox_ψ is recovered.
Let us 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 [33],

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

B. 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 $\mathbb{R}^N$:

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

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:

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

A number of primal-dual algorithms [10], [14]–[16] can be applied to solve Problem (5) by making use of its dual formulation given by:

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

where $(g_j^*)_{1\leqslant j\leqslant J}$ are the Fenchel-Legendre conjugate functions of $(g_j)_{1\leqslant j\leqslant J}$.
Among existing efficient primal-dual approaches [34], we can mention the Dual Block Preconditioned Forward-Backward algorithm recently proposed in [25], presented in Algorithm 1.

Algorithm 1 benefits from the acceleration provided by variable metric methods through the introduction of preconditioning matrices $(B_j)_{1\leqslant j\leqslant J}$. Note that a non-preconditioned version is obtained by setting $(\forall j \in \{1,\ldots,J\})$ $B_j = \|A_j\|^2 I_{M_j}$ 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 $j_n \in \{1,\ldots,J\}$ 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 [34]. Convergence guaranties on both generated primal sequence $(x_n)_{n\in\mathbb{N}}$ and dual sequences $(y_n^j)_{n\in\mathbb{N}*}$ with $j \in \{1,\ldots,J\}$ 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 $K \geqslant J$).

Algorithm 1: Dual Block Preconditioned Forward-Backward

Initialization:

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

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

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

III. Distributed algorithm

Let us ground on the previous algorithm in order to design a distributed (i.e., multi-node) solution to Problem (5). This can be achieved by resorting to a global consensus technique [3], [16], [27], [35] and rewriting the problem in the following form:

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

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 $\mathbb{R}^{NJ}$ whose components $(x^j)_{1 \leqslant j \leqslant J}$ are all equal, and equals the solution $\widehat{x}$ to Problem (5).

A. Local form of consensus

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

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

Examples of vector subspaces $(\Lambda_\ell)_{1\leqslant\ell\leqslant L}$ allowing this condition to be satisfied will be discussed in Section III-C. Each node $j \in \{1,\dots,J\}$ 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 is encompassed by 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 $\{1, 2, 5, 7\}$.

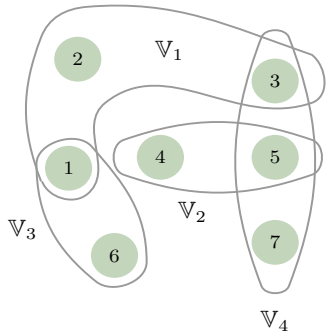


Fig. 1: Connected hypergraph of $J = 7$ nodes and $L = 4$ hyperedges.

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 $\boldsymbol{x} = [(x^1)^\top,\dots,(x^J)^\top]^\top \in \mathbb{R}^{NJ}$:

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

where $\mathrm{i}(\ell,1),\dots,\mathrm{i}(\ell,\kappa_\ell)$ 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

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

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):

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

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 equal to 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:

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

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

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

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

B. Derivation of the proposed algorithm

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

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

Let us 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 $(\forall \ell \in \{1, \ldots, L\})$ $(\forall \gamma \in]0, +\infty[)$,

$$\text{prox}_{\gamma^{-1} g_{J+\ell}}(\gamma^{-1} \cdot) = \gamma^{-1} \Pi_{\Lambda_\ell}, \tag{17}$$

where Π_{Λ_ℓ} is the linear projector onto the vector space Λ_ℓ.

Hence, by setting

$$(\forall \ell \in \{1, \ldots, L\}) \qquad B_{J+\ell} = \vartheta_\ell^{-1} I_{N\kappa_\ell} \tag{18}$$

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

$$\mathbb{V}_j^* = \big\{(\ell, k) \;\big|\; \ell \in \{1, \ldots, L\},\, k \in \{1, \ldots, \kappa_\ell\} \text{ and } \mathfrak{i}(\ell, k) = j\big\}, \tag{19}$$

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 $n = 0, 1, \dots$

$$
\begin{array}{l}
\gamma_n \in [\epsilon, 2 - \epsilon] \\
j_n \in \{1, \dots, J + L\} \\
\text{If } j_n \leqslant J \\
\quad \left\lfloor
\begin{array}{l}
\widetilde{y}_n^{j_n} = y_n^{j_n} + \gamma_n (B_{j_n})^{-1} A_{j_n} x_n^{j_n} \\
y_{n+1}^{j_n} = \widetilde{y}_n^{j_n} - \gamma_n (B_{j_n})^{-1} \mathrm{prox}_{\gamma_n (B_{j_n})^{-1}, g_{j_n}} \big(\gamma_n^{-1} B_{j_n} \widetilde{y}_n^{j_n} \big) \\
y_{n+1}^{j} = y_n^{j}, \quad j \in \{1, \dots, J\} \backslash \{j_n\} \\
z_{n+1}^{\ell} = z_n^{\ell}, \quad \ell \in \{1, \dots, L\} \\
x_{n+1}^{j_n} = x_n^{j_n} - \Omega_{j_n}^{-1} A_{j_n}^\top (y_{n+1}^{j_n} - y_n^{j_n}) \\
x_{n+1}^{j} = x_n^{j}, \quad j \in \{1, \dots, J\} \backslash \{j_n\}
\end{array}
\right. \\
\quad \text{else} \\
\quad \left\lfloor
\begin{array}{l}
\ell_n = j_n - J \\
\widetilde{z}_n^{\ell_n} = z_n^{\ell_n} + \gamma_n \vartheta_{\ell_n} (x_n^j)_{j \in \mathbb{V}_{\ell_n}} \\
z_{n+1}^{\ell_n} = \widetilde{z}_n^{\ell_n} - \Pi_{\Lambda_{\ell_n}} (\widetilde{z}_n^{\ell_n}) \\
z_{n+1}^{\ell} = z_n^{\ell}, \quad \ell \in \{1, \dots, L\} \backslash \{\ell_n\} \\
y_{n+1}^{j} = y_n^{j}, \quad j \in \{1, \dots, J\} \\
\text{For } k = 1, \dots, \kappa_{\ell_n} \\
\quad \left\lfloor \; x_{n+1}^{\mathsf{i}(\ell_n,k)} = x_n^{\mathsf{i}(\ell_n,k)} - \Omega_{\mathsf{i}(\ell_n,k)}^{-1} (z_{n+1}^{\ell_n,k} - z_n^{\ell_n,k}) \right. \\
x_{n+1}^{j} = x_n^{j}, \quad j \notin \mathbb{V}_{\ell_n}.
\end{array}
\right.
\end{array}
\tag{20}
$$

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

$$
\boldsymbol{x}_n = [(x_n^1)^\top, \dots, (x_n^J)^\top]^\top = \boldsymbol{D} \boldsymbol{x}'_n, \tag{21}
$$

$$
z_n^\ell = y_n^{J+\ell} \in \mathbb{R}^{N\kappa_\ell}, \quad \widetilde{z}_n^\ell = \widetilde{y}_n^{J+\ell} \in \mathbb{R}^{N\kappa_\ell}. \tag{22}
$$

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

$$\begin{aligned}\Pi_{\Lambda_{\ell_n}}(z_{n+1}^{\ell_n}) &= \Pi_{\Lambda_{\ell_n}}\big(\widetilde{z}_n^{\ell_n} - \Pi_{\Lambda_{\ell_n}}(\widetilde{z}_n^{\ell_n})\big) \\ &= \Pi_{\Lambda_{\ell_n}}\big(\widetilde{z}_n^{\ell_n}\big) - \Pi_{\Lambda_{\ell_n}}\big(\Pi_{\Lambda_{\ell_n}}(\widetilde{z}_n^{\ell_n})\big) \\ &= 0. \end{aligned} \tag{23}$$

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

$$(\forall n \in \mathbb{N})(\forall \ell \in \{1,\ldots,L\}) \quad \Pi_{\Lambda_\ell}(z_n^{\ell}) = 0, \tag{24}$$

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

$$\Pi_{\Lambda_{\ell_n}}(\widetilde{z}_n^{\ell}) = \gamma_n \vartheta_{\ell_n} \Pi_{\Lambda_{\ell_n}}\big((x_n^j)_{j\in\mathbb{V}_{\ell_n}}\big), \tag{25}$$

which implies that

$$z_{n+1}^{\ell_n} - z_n^{\ell_n} = \gamma_n \vartheta_{\ell_n}\big((x_n^j)_{j\in\mathbb{V}_{\ell_n}} - \Pi_{\Lambda_{\ell_n}}\big((x_n^j)_{j\in\mathbb{V}_{\ell_n}}\big)\big). \tag{26}$$

The second part of iteration n of (20) 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 Algorithm 2 is composed of two main parts:

- First a local optimization part (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 the main 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.
- The second part of Algorithm 2 is 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 the matrices $(A_j)_{1\leqslant j\leqslant J}$, which is critical when these matrices do not have a simple structure and are of very large size. Note that the proposed approach is quite different from the ones developed in [27], [28] since it does not assume a random sweeping rule for the block index selection, and its convergence analysis does not rely on the nonexpansiveness property of the involved operators.

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 $B_j \geq A_j \Omega_j^{-1} A_j^\top, \quad j \in \{1,\dots,J\}$
$\vartheta_\ell = \min_{j \in \mathbb{V}_\ell} \|\Omega_j\|, \quad \ell \in \{1,\dots,L\}$
$\epsilon \in]0,1]$
$y_0^j \in \mathbb{R}^{M_j}, x_0^j = \widetilde{x} - \Omega_j^{-1} A_j^\top y_0^j, \quad j \in \{1,\dots,J\}.$
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:
    $\widetilde{y}_n^{j_n} = y_n^{j_n} + \gamma_n (B_{j_n})^{-1} A_{j_n} x_n^{j_n}$
    $y_{n+1}^{j_n} = \widetilde{y}_n^{j_n} - \gamma_n (B_{j_n})^{-1} \mathrm{prox}_{\gamma_n (B_{j_n})^{-1}, g_{j_n}} \big(\gamma_n^{-1} B_{j_n} \widetilde{y}_n^{j_n}\big)$
    $y_{n+1}^{j} = y_n^{j}, \quad j \in \{1,\dots,J\} \backslash \{j_n\}$
    $x_{n+1}^{j_n} = x_n^{j_n} - \Omega_{j_n}^{-1} A_{j_n}^\top (y_{n+1}^{j_n} - y_n^{j_n})$
    $x_{n+1}^{j} = x_n^{j}, \quad j \in \{1,\dots,J\} \backslash \{j_n\}$
  else
    Projection:
    $\ell_n = j_n - J$
    $y_{n+1}^{j} = y_n^{j}, \quad j \in \{1,\dots,J\}$
    $p_n^{\ell_n} = \Pi_{\Lambda_{\ell_n}} \big((x_n^j)_{j \in \mathbb{V}_{\ell_n}}\big)$
    for $k = 1, \dots, \kappa_{\ell_n}$ **do**
      $x_{n+1}^{\mathrm{i}(\ell_n,k)} = x_n^{\mathrm{i}(\ell_n,k)} + \gamma_n \vartheta_{\ell_n} \Omega_{\mathrm{i}(\ell_n,k)}^{-1} (p_n^{\ell_n,k} - x_n^{\mathrm{i}(\ell_n,k)})$
    end
    $x_{n+1}^{j} = x_n^{j}, \quad j \notin \mathbb{V}_{\ell_n}.$
  end
end

C. Consensus choice

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):

$$(\forall \ell \in \{1,\dots,L\}) \qquad \Lambda_\ell = \left\{ \begin{bmatrix} v^{\ell,1} \\ \vdots \\ v^{\ell,\kappa_\ell} \end{bmatrix} \in \mathbb{R}^{N\kappa_\ell} \mid v^{\ell,1} = \ldots = v^{\ell,\kappa_\ell} \right\}. \tag{27}$$

Note that (8), (10) and (27) 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

$$\big(\forall (v^{\ell,k})_{1 \leqslant k \leqslant \kappa_\ell} \in \mathbb{R}^{N\kappa_\ell}\big) \qquad \Pi_{\Lambda_\ell}\big((v^{\ell,k})_{1 \leqslant k \leqslant \kappa_\ell}\big) = [(\overline{v}^\ell)^\top, \dots, (\overline{v}^\ell)^\top]^\top, \tag{28}$$

where

$$\overline{v}^{\ell} = \mathrm{mean}\left((v^{\ell,k})_{1\leqslant k\leqslant \kappa_\ell}\right) \tag{29}$$

and $\mathrm{mean}(\cdot)$ designates the arithmetic mean operation (i.e. $\mathrm{mean}\left((v^{\ell,k})_{1\leqslant k\leqslant \kappa_\ell}\right) = \kappa_\ell^{-1}\sum_{k=1}^{\kappa_\ell} v^{\ell,k}$).

In addition, Condition (9) is met by simply choosing $(\forall j \in \{1,\ldots,J\})\ \Omega_j = \omega_j I_N$, where $(\omega_j)_{1\leqslant j\leqslant J} \in]0,1]^J$ are such that $\sum_{j=1}^J \omega_j = 1$

These simplification lead to the following modifications of lines 22-25 in Algorithm 2:

$$\begin{array}{l} \overline{x}_n^{\ell_n} = \mathrm{mean}\left((x_n^j)_{j\in\mathbb{V}_{\ell_n}}\right) \\ \text{For } k = 1,\ldots,\kappa_{\ell_n} \\ \left\lfloor\ x_{n+1}^{\mathrm{i}(\ell_n,k)} = x_n^{\mathrm{i}(\ell_n,k)} + \gamma_n \vartheta_{\ell_n} \omega_{\mathrm{i}(\ell_n,k)}^{-1} (\overline{x}_n^{\ell_n} - x_n^{\mathrm{i}(\ell_n,k)}). \end{array} \tag{30}$$

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

$$(\forall j \in \{1,\ldots,J\})\big(\forall x^j = ([x^j]_t)_{1\leqslant t\leqslant T} \in \mathbb{R}^N\big) \qquad A_j x^j = \sum_{t\in\mathbb{T}_j} \mathcal{A}_{j,t}[x^j]_t \tag{31}$$

where, for every $j \in \{1,\ldots,J\}$, $[x^j]_t$ is a vector corresponding to a block of data of dimension L, T is the overall number of blocks (i.e., $N = TL$), and $\mathbb{T}_j \subset \{1,\ldots,T\}$ 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 $\Omega_j = \mathrm{Diag}\,(\Omega_{j,1},\ldots,\Omega_{j,T})$ where, for every $t \in \{1,\ldots,T\}$, $\Omega_{j,t}$ 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 $(\forall j \in \{1,\ldots,J\})\ \mathbb{T}_j \neq \varnothing$ and $\bigcup_{j=1}^J \mathbb{T}_j = \{1,\ldots,T\}$.

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 $\ell \in \{1,\ldots,L\}$) is defined as

$$\begin{aligned} (x^j)_{1\leqslant j\leqslant J} \in \Lambda \ \Leftrightarrow\ & (\forall (j,j') \in \{1,\ldots,J\}^2) \\ & (\forall t \in \mathbb{T}_j \cap \mathbb{T}_{j'}) \quad [x^j]_t = [x^{j'}]_t \end{aligned} \tag{32}$$

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

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

$$([x^j]_t)_{1\leqslant j\leqslant J, t\in\mathbb{T}_j} \mapsto \sum_{j=1}^{J} g_j\Big(\sum_{t\in\mathbb{T}_j} \mathcal{A}_{j,t}[x^j]_t\Big) + \frac{1}{2}\sum_{j=1}^{J}\sum_{t\in\mathbb{T}_j} \|[x^j]_t - [\widetilde{x}]_t\|^2_{\Omega_{j,t}} \tag{33}$$

  subject to Constraint (32);

- the unconstrained minimization of the function

$$([x^j]_t)_{1\leqslant j\leqslant J, t\notin\mathbb{T}_j} \mapsto \sum_{j=1}^{J}\sum_{t\notin\mathbb{T}_j} \|[x^j]_t - [\widetilde{x}]_t\|^2_{\Omega_{j,t}}. \tag{34}$$

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

$$(\forall t\in\{1,\ldots,T\}) \qquad \mathbb{T}_t^* = \big\{j\in\{1,\ldots,J\} \;\big|\; t\in\mathbb{T}_j\big\}, \tag{35}$$

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 $g_j(A_jx)$ 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 $(\forall j\in\{1,\ldots,J\})\ \mathbb{T}_j = \{1,\ldots,T\}$, the consensus solution described in Section III-C1 is recovered. It must be however pointed out that, in general, to have the equivalence between the minimization of (33) subject to Constraint (32) and the resolution of Problem (5), the following condition has to be substituted for (9):

$$(\forall t\in\{1,\ldots,T\}) \qquad \sum_{j\in\mathbb{T}_t^*}\Omega_{j,t} = I_L. \tag{36}$$

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

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

IV. A USEFUL SPECIAL CASE

Let us consider the case when $C\leqslant J$ processing units are available. In the remainder of the paper, to simplify our presentation, we will restrict our attention to a case of practical interest for the video application described in

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

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

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

$\epsilon \in]0,1]$

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

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:

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

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

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

    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**

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

      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

Section V 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 $[x^j]_t$ 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 overlap is allowed between block indices shared with the preceding hyperedge and the following one.

A. *Form of the algorithm*

An interesting 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. 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 28), 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 $\mathbb{V}_1 = \{1,\ldots,J\}$. Unlikely, the global synchronization is mandatory although it has not to be performed at each iteration but only in a quasi-cyclic manner.

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 the one of the parallel dual forward-backward algorithm in [34].

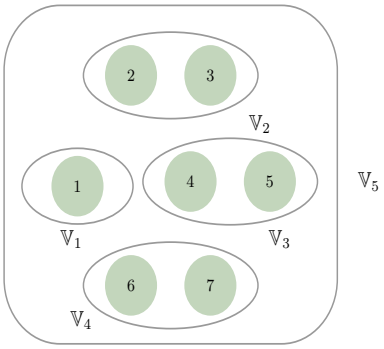


Fig. 2: Hypergraph of $J = 7$ nodes, $C = 4$ computing units and $L = 5$ hyperedges.

B. *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 3 (top) shows an illustrative example of $C = 4$ computing units based on the hypergraph defined in Figure 2.

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

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

$\epsilon \in]0,1]$

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

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**

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

      end

    end

  end

end

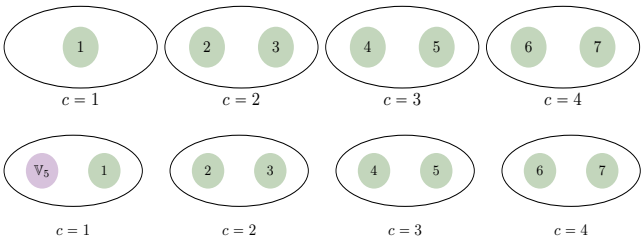


Fig. 3: (top) Partitioning of $J = 7$ nodes and $L = 5$ hyperedges on $C = 4$ computing units. (bottom) Partitioning of $J = 7$ nodes and $L = 1$ hyperedge on $C = 4$ computing systems.

As we have seen, each computing unit $c \in \{1, \ldots, C\}$ 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 to be performed. This task could be assigned to one of the computing unit, say the first one, as modelled in Figure 3 (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 would consist of distributing the operations performed on line 35 of Algorithm 4 over the different computing units. For this purpose, let us 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

$$\begin{aligned}[\overline{x}_n]_t &= \frac{1}{|\mathbb{T}^*_t|}\sum_{j\in\mathbb{T}^*_t}[x^j_{n+1/2}]_t \\ &= \frac{1}{|\mathbb{T}^*_t|}\big([s_{n,c-1}]_t + [s_{n,c}]_t\big),\end{aligned} \tag{37}$$

where

$$[s_{n,c-1}]_t = \sum_{j\in\mathbb{V}_{c-1}\cap\mathbb{T}^*_t}[x^j_{n+1/2}]_t, \tag{38}$$

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 $\mathbb{T}_{\mathbb{V}_{c+1}}$,

$$[\overline{x}_n]_t = \operatorname{mean}\big(([x^j_{n+1/2}]_t)_{j\in\mathbb{V}_c\cap\mathbb{T}^*_t}\big) = \frac{[s_{n,c}]_t}{|\mathbb{T}^*_t|}, \tag{39}$$

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 4, the synchronization workflow is summarized, while, in Algorithm 5, a more detailed account of the whole process

is given.

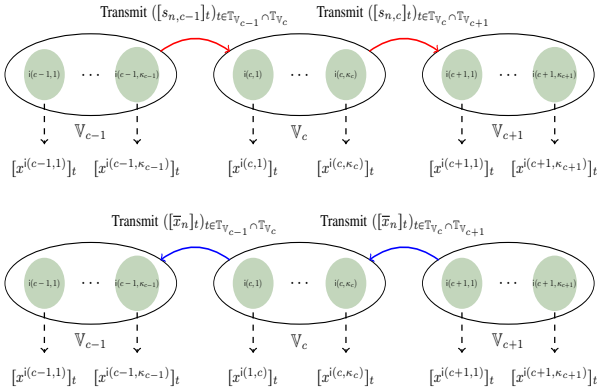


Fig. 4: Global synchronisation process: Transmission of local summations to the next computing unit (top) ; Transmission of averaged blocks to the previous computing unit (bottom).

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 ability 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 $(x_{n+1}^j)_{j\in\mathbb{V}_c} = (x_n^j)_{j\in\mathbb{V}_c}$. 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 be devised. For example, another choice would consist in setting $L = 2C-1$ and $(\forall c \in \{1,\dots,C-1\})\ \mathbb{V}_{C+c} = \mathbb{V}_c \cup \mathbb{V}_{c+1}$. Then, each node $c \in \{1,\dots,C-1\}$ 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.

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

V. Application to video denoising

A. Observation model

In this section, we provide a validation of the proposed distributed algorithm for denoising video sequences. The original sequence $\overline{x} = ([\overline{x}]_t)_{1 \leqslant t \leqslant T} \in \mathbb{R}^{TL}$ 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 $y = ([y]_t)_{1 \leqslant t \leqslant T} \in \mathbb{R}^{TL}$ to the sought sequence $\overline{x}$ with $TL = N$ is given by

$$(\forall t \in \{1, \ldots, T\}) \quad [y]_t = [\overline{x}]_t + [w]_t, \tag{40}$$

where $([w]_t)_{1 \leqslant t \leqslant T} \in \mathbb{R}^{TL}$ represents an additive zero-mean white Gaussian noise. An estimate of the unknown video can be inferred by solving Problem (5) where $J = T$ and $\widetilde{x} = y$. The last quadratic term in (5) is a least squares data fidelity term ensuring compliance with model (40), 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 $t \in \{2, \ldots, T-1\}$, the linear operator A_t extracts the current frame x_t and its neighbors (x_{t-1}, x_{t+1}) as shown by:

$$[[x]_1 \ldots [x]_{t-1} [x]_t [x]_{t+1} \ldots [x]_T] \xrightarrow{A_t} [[x]_{t-1} [x]_t [x]_{t+1}] \tag{41}$$

The linear operators $(A_t)_{1 \leqslant t \leqslant T}$ thus have the block sparse structure expressed by (31) with

$$(\forall t \in \{1, \ldots, T\}) \quad \mathbb{T}_t = \big\{ \max\{t-1, 1\}, t, \min\{t+1, T\} \big\} \tag{42}$$

and

$$\mathcal{A}_{1,1} = \begin{bmatrix} I_L & 0 \end{bmatrix}^\top, \quad \mathcal{A}_{1,2} = \begin{bmatrix} 0 & I_L \end{bmatrix}^\top, \tag{43}$$

$$(\forall t \in \{2, \ldots, T-1\}) \quad \mathcal{A}_{t,t-1} = \begin{bmatrix} I_L & 0 & 0 \end{bmatrix}^\top \tag{44}$$

$$\mathcal{A}_{t,t} = \begin{bmatrix} 0 & I_L & 0 \end{bmatrix}^\top \tag{45}$$

$$\mathcal{A}_{t,t+1} = \begin{bmatrix} 0 & 0 & I_L \end{bmatrix}^\top \tag{46}$$

$$\mathcal{A}_{T,T-1} = \begin{bmatrix} I_L & 0 \end{bmatrix}^\top, \quad \mathcal{A}_{T,T} = \begin{bmatrix} 0 & I_L \end{bmatrix}^\top. \tag{47}$$

For every $t \in \{1, \ldots T\}$, each regularization function $g_t \colon \mathbb{R}^{M_t} \to [0, +\infty[$ is convex, proper, lower semi-continuous and such that

$$M_t = \begin{cases} 3L & \text{if } t \neq 1 \text{ and } t \neq T \\ 2L & \text{otherwise,} \end{cases} \tag{48}$$

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

$$g_t\left(([x]_{t'})_{t' \in \mathbb{T}_t}\right) = \eta \ \mathrm{tgv}([x]_t) + \iota_{[x_{\min}, x_{\max}]^L}([x]_t) + h_t\left(([x]_{t'})_{t' \in \mathbb{T}_t}\right), \tag{49}$$

where "tgv" denotes the *Total Generalized Variation* regularization from [36], defined as

$$(\forall z \in \mathbb{R}^L) \quad \mathrm{tgv}(z) = \min_{q \in \mathbb{R}^{2L}} \ \alpha_0 \, \chi_2 \left(Dz - q \right) + \alpha_1 \, \chi_3 \left(\mathsf{G} q \right), \tag{50}$$

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:

$$D = \begin{bmatrix} \nabla_{\mathsf{H}} \\ \nabla_{\mathsf{V}} \end{bmatrix}, \quad \text{with} \quad \nabla_{\mathsf{H}} \in \mathbb{R}^{L \times L}, \quad \nabla_{\mathsf{V}} \in \mathbb{R}^{L \times L}, \tag{51}$$

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

$$\mathsf{G} = \begin{bmatrix} \nabla_{\mathsf{H}} & \nabla_{\mathsf{V}} & 0 \\ 0 & \nabla_{\mathsf{H}} & \nabla_{\mathsf{V}} \end{bmatrix}^\top, \tag{52}$$

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

$$\left(\forall (z_1, \ldots, z_q) \in (\mathbb{R}^L)^q \right) \quad \chi_q(z_1, \ldots, z_q) = \sum_{k=1}^{L} \sqrt{(z_{1,k})^2 + \cdots + (z_{q,k})^2}. \tag{53}$$

The indicator function $\iota_{[x_{\min}, x_{\max}]^L}$ in (49) 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

$$h_t \left(([x]_{t'})_{t' \in \mathbb{T}_t} \right) = \begin{cases} \beta_{t-1,t} \, \chi_1([x]_t - \mathcal{M}_{t-1 \to t} \, [x]_{t-1}) \\ \qquad + \beta_{t+1,t} \, \chi_1([x]_t - \mathcal{M}_{t+1 \to t} \, [x]_{t+1}) \\ \qquad\qquad \text{if } t \neq 1 \text{ and } t \neq T \\ \beta_{2,1} \, \chi_1([x]_1 - \mathcal{M}_{2 \to 1} \, [x]_2) \\ \qquad\qquad \text{if } t = 1 \\ \beta_{T-1,T} \, \chi_1([x]_T - \mathcal{M}_{T-1 \to T} \, [x]_{T-1}) \\ \qquad\qquad \text{if } t = T, \end{cases} \tag{54}$$

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]. 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 set optimized by grid-search so as to achieve the best denoising performance.

B. 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 the

same number of nodes, i.e., $(\forall c \in \{1,\dots,C\})\ \kappa_c = \kappa$ (with $T = \kappa C$). The associated hyperedges are then given by

$$(\forall c \in \{1,\dots,C\}) \qquad \mathbb{V}_c = \{(c-1)\kappa + 1, \dots, c\kappa\}. \tag{55}$$

Note that, since

$$(\forall c \in \{1,\dots,C\}) \qquad \mathbb{T}_{\mathbb{V}_c} = \begin{cases} \max\{(c-1)\kappa, 1\}, \dots, \\ \qquad\qquad \min\{c\kappa + 1, T\} \end{cases} \tag{56}$$

$$\Rightarrow \quad (\forall c \in \{1,\dots,C-1\}) \qquad \mathbb{T}_{\mathbb{V}_c} \cap \mathbb{T}_{\mathbb{V}_{c+1}} = \{c\kappa, c\kappa + 1\}, \tag{57}$$

Assumption 4.1 holds provided that $\kappa > 1$.

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 IV-B. In our case, for every $t \in \{1,\dots,T\}$,

$$\mathbb{T}_t^* = \mathbb{T}_t. \tag{58}$$

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 $t = (c-1)\kappa$ (resp. $c < C$ and $t = c\kappa + 1$)), 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 (57), only variables $[s_{n,c}]_{c\kappa}$ and $[s_{n,c}]_{c\kappa+1}$ need to be transmitted from computing unit $c \neq C$ to computing unit $c+1$, which in return sends back the updated averages $[\overline{x}_n]_{c\kappa}$ and $[\overline{x}_n]_{c\kappa+1}$. This workflow is illustrated in Figures 5 and 6 by an example showing two computing units handling $\kappa = 3$ nodes.

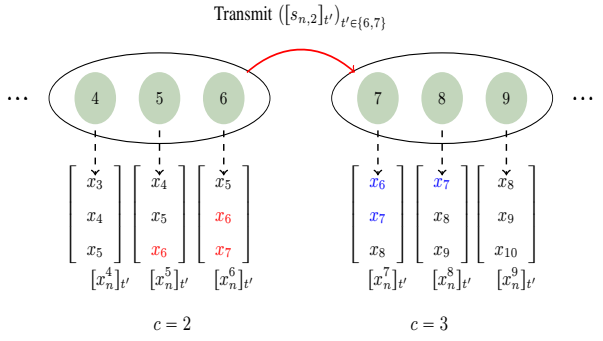


Fig. 5: Transmission of local sums $([s_{n,2}]_{t'})_{t' \in \{6,7\}}$ shared between $\mathbb{T}_{\mathbb{V}_2} = \{3,4,5,6,7\}$ and $\mathbb{T}_{\mathbb{V}_3} = \{6,7,8,9,10\}$ from computing unit $c = 2$ to computing unit $c = 3$.

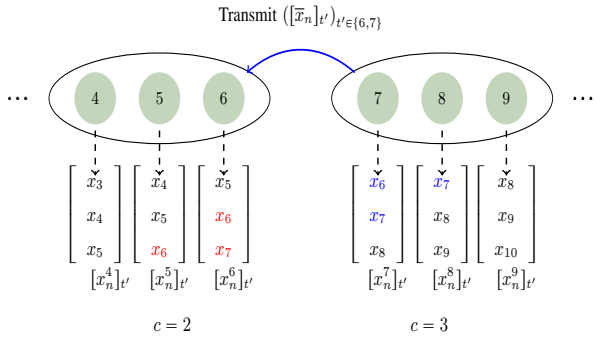


Fig. 6: Transmission of averaged images $([\overline{x}_n]_{t'})_{t'\in\{6,7\}}$ from computing unit $c = 3$ to computing unit $c = 2$.

In our simulations, the global synchronizations are activated every 4 iterations. This synchronization frequency was chosen in order to achieve a good 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 $\frac{1}{|\mathbb{T}_t^*|}$.

C. Simulation results

The performance of the proposed denoising method are evaluated on the standard test video sequences **Foreman**, **Claire** and **Irene** with $T = 72$ frames. These frames are of size 348 $\times$ 284 for **Foreman** sequence, 300 $\times$ 278 and 352 $\times$ 288 of **Claire** and **Irene** respectively, hence $N = 7115904$ (resp. $N = 6004800$ and $N = 7299072$). The degraded videos are obtained by adding zero-mean white Gaussian noise to the original video sequences, resulting in an initial SNR (signal-to-noise ratio) of 24.41 dB, 24.77 dB and 25.51 dB for the three sequences respectively. We apply our algorithm only on the luminance channel, while the chrominance is restored with a median filter. Our method is implemented with Julia-0.4.6 and a *Message Passing Interface* (MPI) wrapper for managing communication between cores [37], [38]. We use a multi-core architecture using 2 Intel(R) Xeon(R) E5-2670 v3 CPU @ 2.3 GHz processors, each with 12 cores, hence $C = 24$. The experiments are run using 60 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. The images composing the video sequences are partitioned in groups of equal size κ processed by the computing units, thereby we consider the cases when $C \in \{1, 2, 3, 4, 6, 8, 9, 12, 18, 24\}$ cores are employed, as shown in Table I.

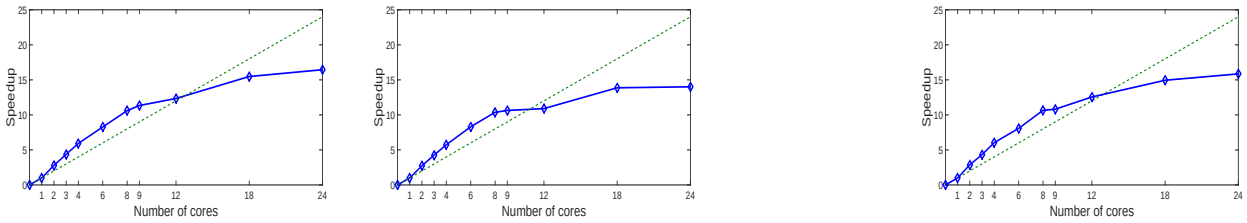
TABLE I: Investigated simulation scenarios and the number of images per core in each case.

Number of cores C	1	2	3	4	6	8	9	12	18	24
Number of images per core κ	72	36	24	18	12	9	8	6	4	3

Figure 7 shows the speedup in execution time with respect to the number of cores, which is estimated as follows:

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

The execution time with one core is equal to 107003 s, 84247 s and 115711 s for **Foreman**, **Claire** and **Irene** sequences respectively.



(a) **Foreman** sequence. (b) **Claire** sequence. (c) **Irene** sequence.

Fig. 7: Speedup with respect to the number of used cores: proposed method (solid, blue, diamond), linear speedup (dashed, green).

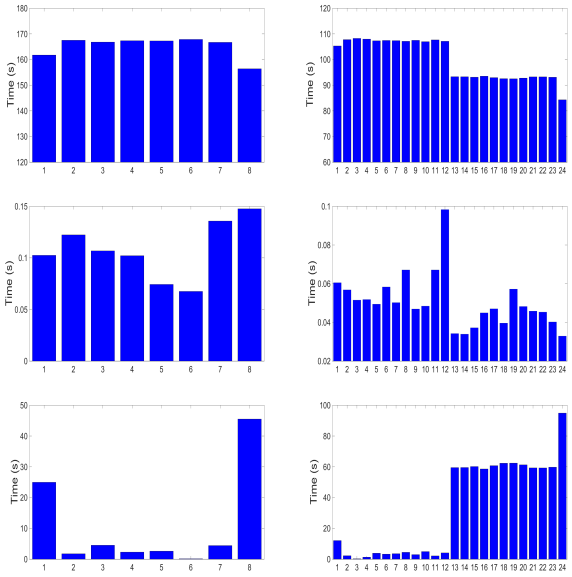
Figure 7 shows that the speedup increases super linearly as we increase the number of cores from 1 to 9. Indeed, when a small number of cores are used, the dataset cannot be stored in the cache memory, due to its large size. Hence, a significant amount of time is spent in RAM access [39]. By increasing the number of cores, the data seem to fit better in the cache size, which reduces the RAM access time and consequently the global execution time despite the communication overhead. However, as the number of core exceeds 9, a saturation effect is observed (in agreement with Amdahl's law [40]) .

In order to investigate this behavior, we display in Figure 8 the execution times per core on the **Foreman** sequence, for the three main steps of Algorithm 5. Namely, the local optimization, local synchronization, and global synchronization when either $C = 8$ or $C = 24$ cores are used. As expected we observe a significant reduction of the execution time for the local optimization step when going from 8 to 24 cores, but the gain factor is less than 3, although the computations are then performed independently on each core. The average execution time for the local synchronization step is also reduced as the number of images handled by each core decreases. One can finally observe that the global communication overhead increases as a larger number of cores is used. This behavior appears to be consistent, however it can be noticed on Figure 8(b) that the second set of cores (13 to 24) is much slower than the first one, which is detrimental to the global synchronization process. This seems to point out hardware limitations of the Intel-based two-processor computer architecture that we use.

Our method achieves satisfactory restoration results with an improvement of 7.6 dB for **Foreman**, 9 dB for **Claire** and 5.46 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. Figures 9 and 10 show some frames illustrative of the degraded and restored sequences. These illustrate the good visual quality of the performed denoising.

VI. Conclusion

This paper 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.



(a) 8 cores (b) 24 cores

Fig. 8: Execution time of Algorithm 5 steps: local optimization (top), local synchronization (middle), global synchronization (bottom).

Fig. 9: **Foreman** sequence: Input degraded images (top) initial SNR = 24.41 dB, associated restored images (bottom) final SNR = 32.04 dB.

We mainly focused on an instance of the proposed approach for which we proposed an asynchronous implementation, assuming that a given number of computing units is available. Although our distributed algorithm can be applied to a wide range of inverse 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

Fig. 10: **Irene** sequence: Input degraded images (top) initial SNR = 25.51 dB, associated restored images (bottom) final SNR = 30.97 dB.

cores. An acceleration of about 15 was reached with a standard two-processor 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.

REFERENCES

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

[2] N. Parikh and S. Boyd, "Proximal algorithms," *Found. Trends Optim.*, vol. 1, no. 3, pp. 127–239, Jan. 2014.

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

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

[5] L. Condat, "A primal-dual splitting method for convex optimization involving lipschitzian, proximable and linear composite terms," *J. Optim. Theory App.*, vol. 158, no. 2, pp. 460–479, Aug. 2013.

[6] S. R. Becker and P. L. Combettes, "An algorithm for splitting parallel sums of linearly composed monotone operators, with applications to signal recovery," *J. Nonlinear Convex Anal.*, vol. 15, no. 1, pp. 137–159, Jan. 2014.

[7] C. Couprie, L. Grady, L. Najman, J.-C. Pesquet, and H. Talbot, "Dual constrained tv-based regularization on graphs," *SIAM J. Imaging Sci.*, vol. 6, pp. 1246–1273, Jul. 2013.

[8] A. Jezierska, E. Chouzenoux, J.-C. Pesquet, and 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, 25-30 Mar. 2012, pp. 1085–1088.

[9] A. Onose, R. E. Carrillo, A. Repetti, J. D. McEwen, J.-T. Thiran, J.-C. Pesquet, and Y. Wiaux, "Scalable splitting algorithms for big-data interferometric imaging in the SKA era," *Monthly Notices of the Royal Astronomical Society*, vol. 462, no. 4, pp. 4314–4335, 2016.

[10] A. Chambolle and T. Pock, "A first-order primal-dual algorithm for convex problems with applications to imaging," *J. Math. Imag. Vision*, vol. 40, no. 1, pp. 120–145, May 2010.

[11] P. L. Combettes, L. Condat, J.-C. Pesquet, and 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, 27-30 Oct. 2014, pp. 4141–4145.

[12] R. I. Boţ and C. Hendrich, "Convergence analysis for a primal-dual monotone + skew splitting algorithm with applications to total variation minimization," *J. Math. Imaging Vision*, vol. 49, no. 3, pp. 551–568, Jul. 2014.

[13] P. L. Combettes and 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.*, vol. 20, no. 2, pp. 307–330, Jun. 2012.

[14] R. I. Boţ and 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.*, vol. 23, no. 4, pp. 2541–2565, Dec. 2013.

[15] P. L. Combettes and J.-C. Pesquet, "Stochastic quasi-Fejér block-coordinate fixed point iterations with random sweeping," *SIAM J. Optim.*, vol. 25, no. 2, pp. 1221–1248, Jul. 2015.

[16] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Machine Learn.*, vol. 8, no. 1, pp. 1–122, Jan. 2011.

[17] L. M. Briceño Arias and P. L. Combettes, "A monotone+skew splitting model for composite monotone inclusions in duality," *SIAM J. Optim.*, vol. 21, no. 4, pp. 1230–1250, Oct. 2011.

[18] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem, "Explicit convergence rate of a distributed alternating direction method of multipliers," *IEEE Trans. Autom. Control*, vol. 61, no. 4, pp. 892–904, Apr. 2016.

[19] D. Davis, "Convergence rate analysis of primal-dual splitting schemes," *SIAM J. Optim.*, vol. 25, no. 3, pp. 1912–1943, Sep. 2015.

[20] L. Rosasco, S. Villa, and B. C. Vũ, "A first-order stochastic primal-dual algorithm with correction step," *Numer. Funct. Anal. Optim.*, vol. 38, no. 5, pp. 602–626, Apr. 2017.

[21] S. Shalev-Shwartz and T. Zhang, "Stochastic dual coordinate ascent methods for regularized loss minimization," *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 567–599, Feb. 2013.

[22] P. Bianchi, W. Hachem, and F. Iutzeler, "A coordinate descent primal-dual algorithm and application to distributed asynchronous optimization," *IEEE Trans. Autom. Control*, vol. 61, no. 10, pp. 2947–2957, Oct. 2016.

[23] Z. Qu, P. Richtárik, and T. Zhang, "Randomized dual coordinate ascent with arbitrary sampling," in *Adv. Neural Inf. Process. Syst. (NIPS 2015)*, Montréal, Canada, 7-12 Dec. 2015, pp. 865–873.

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

[25] F. Abboud, E. Chouzenoux, J.-C. Pesquet, J.-H. Chenot, and L. Laborelli, "Dual block coordinate forward-backward algorithm with application to deconvolution and deinterlacing of video sequences," *J. Math. Imaging Vision*, vol. 59, no. 3, pp. 415–431, Nov. 2017.

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

[27] J.-C. Pesquet and A. Repetti, "A class of randomized primal-dual algorithms for distributed optimization," *J. Nonlinear Convex Anal.*, vol. 16, no. 12, pp. 2453–2490, Dec. 2015.

[28] P. Richtárik and M. Takác, "Distributed coordinate descent method for learning with big data," *J. Mach. Learn. Res.*, vol. 17, no. 75, pp. 1–25, Feb. 2016.

[29] 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 App.*, vol. 162, no. 1, pp. 107–132, Jul. 2014.

[30] ——, "A block coordinate variable metric forward-backward algorithm," *J. Global Optim.*, vol. 66, no. 3, pp. 457–485, Nov. 2016.

[31] S. Becker and J. Fadili, "A quasi-Newton proximal splitting method," in *Adv. Neural Inf. Process. Syst. (NIPS 2012)*, Lake Tahoe, Nevada, 3-8 Dec. 2012, pp. 2627–2635.

[32] J. J. Moreau, "Proximité et dualité dans un espace hilbertien," *Bull. Soc. Math. France*, vol. 93, pp. 273–299, 1965.

[33] P. L. Combettes and B. C. Vũ, "Variable metric forward-backward splitting with applications to monotone inclusions in duality," *Optimization*, vol. 63, no. 9, pp. 1289–1318, Sep. 2014.

[34] P. L. Combettes, D. Dũng, and B. C. Vũ, "Proximity for sums of composite functions," *J. Math. Anal. Appl.*, vol. 380, no. 2, pp. 680–688, Aug. 2011.

[35] N. Pustelnik, C. Chaux, and J.-C. Pesquet, "Parallel proximal algorithm for image restoration using hybrid regularization," *IEEE Trans. Image Process.*, vol. 20, no. 9, pp. 2450–2462, Sep. 2011.

[36] K. Bredies, K. Kunisch, and T. Pock, "Total generalized variation," *SIAM J. Imaging Sci.*, vol. 3, no. 3, pp. 492–526, Sep. 2010.

[37] M. P. I. Forum, "MPI: A Message-Passing Interface standard," Tech. Rep., 1994.

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

[39] D. Janakiram, M. V. Reddy, A. V. Srinivas, M. A. M. Mohamed, and S. S. Kumar, "GDP: A paradigm for intertask communication in grid computing through distributed pipes," in *Int. Conf. Distrib. Comput. Internet Technol. (ICDCIT 2005)*, Bhubaneswar, India, 22-24 Dec. 2005, pp. 235–241.

[40] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *American Fed. Inform. Process. Soc. (AFIPS 1967)*, Atlantic City, USA, 18-20 Apr. 1967, pp. 483–485.