A Fast Parallel Algorithm for Convolutional Sparse Coding

Erik Skau
CCS-3 / T-5, CCS and T Divisions
Los Alamos National Laboratory
Los Alamos, NM 87544, USA
ewskau@gmail.com

Brendt Wohlberg
T-5, T Division
Los Alamos National Laboratory
Los Alamos, NM 87545, USA
brendt@ieee.org

***Abstract*—The current leading algorithms for convolutional sparse coding are not inherently parallelizable, and therefore are not able to fully exploit modern multi-core architectures. We address this deficiency by developing a new algorithm that partitions the dictionary and the corresponding coefficient maps into groups, solving the main subproblems for all of the groups in parallel. Theoretical complexities and implementational details are discussed and validated with computational experiments, which indicate speed improvements by about a factor of 5, depending on the specific problem.**

***Index Terms*—Convolutional Sparse Representations, Convolutional Sparse Coding, ADMM**

I. Introduction

Sparse representations are widely used in signal processing and computer vision [1]. The representation of a signal $\mathbf{s}$ can be written as $D\mathbf{x} \approx \mathbf{s}$, where matrix or linear operator D is the dictionary, and $\mathbf{x}$ is a vector with only a few non-zero entries. While synthesis of a signal from its sparse representation involves a linear operation, the computation of the representation for a given signal, referred to as sparse coding, involves solving an optimization problem.

A translation-invariant representation is obtained when the dictionary consists of a set of linear filters $\mathbf{d}_m$ and the signal representation can be written as $\sum_m \mathbf{d}_m * \mathbf{x}_m \approx \mathbf{s}$. These *convolutional sparse representations* have attracted increasing interest for computational imaging and image processing applications over the past few years [2], [3], [4], [5], [6], [7], [8]. Although significant progress has been made in developing more efficient algorithms [9], [10], [11], convolutional sparse coding (CSC) remains computationally expensive. The present paper describes a new CSC algorithm that is designed to be highly parallelizable, offering a substantial reduction in computation time on a modern multi-core architecture.

II. Convolutional Sparse Coding

CSC is usually posed as the optimization problem [12]

$$\underset{\{\mathbf{x}_m\}}{\arg\min} \ \frac{1}{2}\Big\|\sum_m \mathbf{d}_m * \mathbf{x}_m - \mathbf{s}\Big\|_2^2 + \lambda \sum_m \|\mathbf{x}_m\|_1 \ , \tag{1}$$

where $\mathbf{s}$ is the signal to be decomposed, $\{\mathbf{d}_m\}$ is a set of M dictionary filters, and $\{\mathbf{x}_m\}$ is a set of M coefficient maps that comprise the sparse representation. A variant of this problem that includes a spatial mask vector $\mathbf{w}$ is useful when careful boundary handling is required, or when the signal $\mathbf{s}$ has missing samples [13], [14]

$$\underset{\{\mathbf{x}_m\}}{\arg\min} \frac{1}{2}\Big\|\mathbf{w} \odot \Big(\sum_m \mathbf{d}_m * \mathbf{x}_m - \mathbf{s}\Big)\Big\|_2^2 + \lambda \sum_m \|\mathbf{x}_m\|_1 \ . \tag{2}$$

Problems (1) and (2) are usually solved via the Alternating Direction Method of Multipliers (ADMM) [15] which provides a general framework for solving problems of the form

$$\underset{\mathbf{x},\mathbf{y}}{\arg\min} \ \ f(\mathbf{x}) + g(\mathbf{y}) \ \text{ such that } \ A\mathbf{x} + B\mathbf{y} = \mathbf{c} \tag{3}$$

by iterating over the updates

$$\mathbf{x}^{(j+1)} = \underset{\mathbf{x}}{\arg\min}\, f(\mathbf{x}) + \frac{\rho}{2}\Big\|A\mathbf{x} + B\mathbf{y}^{(j)} - \mathbf{c} + \mathbf{u}^{(j)}\Big\|_2^2 \ , \tag{4}$$

$$\mathbf{y}^{(j+1)} = \underset{\mathbf{y}}{\arg\min}\, g(\mathbf{y}) + \frac{\rho}{2}\Big\|A\mathbf{x}^{(j+1)} + B\mathbf{y} - \mathbf{c} + \mathbf{u}^{(j)}\Big\|_2^2 , \tag{5}$$

$$\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} + A\mathbf{x}^{(j+1)} + B\mathbf{y}^{(j+1)} - \mathbf{c} \ , \tag{6}$$

where $\mathbf{u}$ is the *dual variable* and $\rho > 0$ is the *penalty parameter* [15].

Both problems (1) and (2) can be converted to equivalent problems of the form (3) by *variable splitting*, which involves introducing auxiliary variables and suitable constraints. Since the usual variable splitting for (1) involves an equality constraint

$$\begin{aligned} &\underset{\{\mathbf{x}_m\},\{\mathbf{y}_m\}}{\arg\min} \frac{1}{2}\Big\|\sum_m \mathbf{d}_m * \mathbf{x}_m - \mathbf{s}\Big\|_2^2 + \lambda \sum_m \|\mathbf{y}_m\|_1 \\ &\text{such that} \quad \mathbf{x}_m = \mathbf{y}_m \ \ \forall m \ , \end{aligned} \tag{7}$$

we will refer to the corresponding ADMM algorithm [11] as ADMM-EQ. Problems (1) and (2) can be solved via an ADMM algorithm based on a different splitting [13]

$$\begin{aligned} &\underset{\{\mathbf{x}_m\},\{\mathbf{y}_0\},\{\mathbf{y}_{1,m}\}}{\arg\min} \frac{1}{2}\big\|\mathbf{w} \odot \mathbf{y}_0\big\|_2^2 + \lambda \sum_m \big\|\mathbf{y}_{1,m}\big\|_1 \ \text{ such that} \\ &\qquad \sum_m \mathbf{d}_m * \mathbf{x}_m - \mathbf{s} = \mathbf{y}_0 \quad \text{and} \quad \mathbf{x}_m = \mathbf{y}_{1,m} \ \ \forall m \ . \end{aligned} \tag{8}$$

Since this splitting is based on the mask decoupling method [16], we will refer to the corresponding ADMM algorithm as ADMM-MD.

The $\mathbf{x}$ step (4) is the most computationally expensive component of the ADMM algorithms for both problems. Unfortunately the solution of this step is not easily parallelizable since it involves a forward Discrete Fourier Transform (DFT), solution of independent linear systems for each frequency, and an inverse DFT [11], [14].

III. Dictionary Partition Algorithm

We propose an alternative variable splitting for these problems, based on a partition of the M dictionary filters into L groups $\{G_l\}_{l \in \{0,\ldots,L-1\}}$ where

$$G_i \cap G_j = \emptyset \ \text{ for } \ i \neq j \quad \text{and} \quad \bigcup_l G_l = \{0, \ldots, M-1\} \ .$$

The corresponding variable splitting into ADMM form of problem (2) is

$$\begin{aligned} &\underset{\{\mathbf{x}_m\},\{\mathbf{y}_{0,l}\},\{\mathbf{y}_{1,m}\}}{\arg\min} \frac{1}{2}\Big\|\mathbf{w} \odot \Big(\sum_l \mathbf{y}_{0,l} - \mathbf{s}\Big)\Big\|_2^2 + \lambda \sum_m \big\|\mathbf{y}_{1,m}\big\|_1 \\ &\text{such that} \quad \sum_{g \in G_l} \mathbf{d}_g * \mathbf{x}_g = \mathbf{y}_{0,l} \ \ \forall l \quad \text{and} \quad \alpha\mathbf{x}_m = \alpha\mathbf{y}_{1,m} \ \ \forall m \ , \end{aligned}$$

where α is a parameter chosen to balance the significance of the two constraints[1]. As in the case of (8), these constraints can be combined into ADMM form (3). We will refer to the corresponding ADMM algorithm as ADMM with dictionary partitioning (ADMM-DP). The steps for this algorithm are

$$\mathbf{x}_{G_l}^{(j+1)} = \underset{\{\mathbf{x}_{g\in G_l}\}}{\operatorname{arg\,min}} \frac{\rho}{2}\Big\| \sum_{g\in G_l} \mathbf{d}_g * \mathbf{x}_g - \mathbf{y}_{0,l}^{(j)} + \mathbf{u}_{0,l}^{(j)} \Big\|_2^2 + \frac{\rho}{2} \sum_{g\in G_l} \Big\| \alpha \mathbf{x}_g - \alpha \mathbf{y}_{1,g}^{(j)} + \mathbf{u}_{1,g}^{(j)} \Big\|_2^2 \tag{9}$$

$$\mathbf{y}_0^{(j+1)} = \underset{\mathbf{y}_0}{\operatorname{arg\,min}} \frac{1}{2}\Big\| \mathbf{w} \odot \Big(\sum_l \mathbf{y}_{0,l} - \mathbf{s} \Big) \Big\|_2^2 + \frac{\rho}{2} \sum_l \Big\| \sum_{g\in G_l} \mathbf{d}_g * \mathbf{x}_g^{(j+1)} - \mathbf{y}_{0,l} + \mathbf{u}_{0,l}^{(j)} \Big\| \tag{10}$$

$$\mathbf{y}_{1,G_l}^{(j+1)} = \underset{\{\mathbf{y}_{1,g\in G_l}\}}{\operatorname{arg\,min}} \lambda \sum_{g\in G_l} \big\| \mathbf{y}_{1,g} \big\|_1 + \frac{\rho}{2} \sum_{g\in G_l} \Big\| \alpha \mathbf{x}_g^{(j+1)} - \alpha \mathbf{y}_{1,g} + \mathbf{u}_{1,g}^{(j)} \Big\|_2^2 \tag{11}$$

$$\mathbf{u}_{0,l}^{(j+1)} = \mathbf{u}_{0,l}^{(j)} + \sum_{g\in G_l} \mathbf{d}_g * \mathbf{x}_g^{(j+1)} - \mathbf{y}_{0_l}^{(j+1)} \tag{12}$$

$$\mathbf{u}_{1,G_l}^{(j+1)} = \mathbf{u}_{1,G_l}^{(j)} + \alpha \mathbf{x}_{G_l}^{(j+1)} - \alpha \mathbf{y}_{1,G_l}^{(j+1)} \;, \tag{13}$$

where the $\mathbf{x}$, $\mathbf{y}_1$, $\mathbf{u}_0$, and $\mathbf{u}_1$ steps can each be broken up into L independent problems that can be computed in parallel. The $\mathbf{y}_0$ step cannot be similarly decomposed.

Each parallel $\mathbf{x}$ step involves solving a linear system. We define $\hat{D}_l$ as the block matrix constructed by horizontal concatenation of the diagonal blocks of the DFT of filters in group G_l, and denote the DFT of variable $\mathbf{z}$ by $\hat{\mathbf{z}}$. The $\mathbf{x}_{G_l}$ step is given by the solution to the linear system

$$(\alpha^2 I + \hat{D}^H \hat{D})\mathbf{x} = \hat{D}^H (\hat{\mathbf{y}}_{0,l} - \hat{\mathbf{u}}_{0,l}) + \alpha^2 \hat{\mathbf{y}}_{1,G_l} - \alpha \hat{\mathbf{u}}_{1,G_l} \;,$$

which can be solved efficiently by exploiting the Sherman-Morrison formula [11]. The $\mathbf{y}_1$ step can be solved as

$$\mathbf{y}_{1,G_l}^{(j+1)} = \mathcal{S}_{\frac{\lambda}{\alpha^2\rho}} (\mathbf{x}_{G_l} + \alpha^{-1} \mathbf{u}_{1,G_l}) \;,$$

where $\mathcal{S}_\gamma(\cdot)$ is the soft thresholding function [17, Sec. 6.5.2]

$$\mathcal{S}_\gamma(\mathbf{u}) = \operatorname{sign}(\mathbf{u}) \odot \max(0, |\mathbf{u}| - \gamma) \;,$$

and the $\mathbf{u}_{0,l}$ and $\mathbf{u}_{1,G_l}$ steps are straightforward.

The $\mathbf{y}_0$ minimization (10) cannot be broken into similar independent problems, and requires closer attention in order to derive an efficient solution. This problem can be reformulated by defining

$$\mathbf{y} = \begin{pmatrix} \mathbf{y}_{0,0} \\ \mathbf{y}_{0,1} \\ \vdots \\ \mathbf{y}_{0,L-1} \end{pmatrix}, \; \mathbf{z} = \begin{pmatrix} \sum_{g\in G_0} \mathbf{d}_g * \mathbf{x}_g^{(j+1)} \\ \sum_{g\in G_1} \mathbf{d}_g * \mathbf{x}_g^{(j+1)} \\ \vdots \\ \sum_{g\in G_{L-1}} \mathbf{d}_g * \mathbf{x}_g^{(j+1)} \end{pmatrix} + \begin{pmatrix} \mathbf{u}_{0,0}^{(j)} \\ \mathbf{u}_{0,1}^{(j)} \\ \vdots \\ \mathbf{u}_{0,L-1}^{(j)} \end{pmatrix},$$

$W = \operatorname{diag}(\mathbf{w})$, and $\mathbb{I} = \begin{pmatrix} I_N & I_N & \dots & I_N \end{pmatrix}$ (L identity matrices concatenated horizontally), so that (10) is equivalent to

$$\underset{\mathbf{y}}{\operatorname{arg\,min}} \; \frac{1}{2} \big\| W\mathbb{I}\mathbf{y} - W\mathbf{s} \big\|_2^2 + \frac{\rho}{2} \big\| \mathbf{y} - \mathbf{z} \big\|_2^2 \;. \tag{14}$$

The solution of this problem is given by the linear system,

$$(\rho I_{LN} + \mathbb{I}^\intercal W^\intercal W \mathbb{I})\mathbf{y} = \mathbb{I}^\intercal W^\intercal W \mathbf{s} + \rho \mathbf{z} \;,$$

[1]A similar parameter is also expected to benefit the ADMM-MD algorithm, but that possibility is not explored here.

TABLE I: Complexity breakdown of ADMM algorithms for M filters with C channels and a signal with C channels and N samples.

	ADMM-EQ	ADMM-MD	ADMM-DP
$\mathbf{x}$ step	$CMN \log N$	$CMN \log N$	$C\frac{M}{L} N \log N$
$\mathbf{y}_0$ step	—	CMN	CLN
$\mathbf{y}/\mathbf{y}_1$ step	MN	MN	$\frac{M}{L} N$
$\mathbf{u}_0$ step	—	CN	CN
$\mathbf{u}/\mathbf{u}_1$ step	MN	MN	$\frac{M}{L} N$

which is a degenerate form of the linear equation for one of the $\mathbf{x}$ steps. Similarly to the $\mathbf{x}$ steps, this system can be solved by exploiting the Sherman-Morrison formula [12, Appendix B].

The $\mathbf{y}_0$ linear system is well conditioned, with a condition number of

$$\kappa(\rho I_{LN} + \mathbb{I}^\intercal W^\intercal W \mathbb{I}) = \frac{\max(\mathbf{w})^2 L}{\rho} + 1 \;,$$

and an inverse of

$$(\rho I_{LN} + \mathbb{I}^\intercal W^\intercal W \mathbb{I})^{-1} = \frac{1}{\rho} I_{LN} - \mathbb{I}^\intercal \left(\frac{W^2}{\rho^2 I_N + \rho L W^2} \right) \mathbb{I} \;,$$

where the division is elementwise. In practice, when W is a mask consisting of $\{0, 1\}$ entries, the condition number simplifies to $\frac{L}{\rho} + 1$. For the CSC problem without a mask the condition number is

$$\kappa(\rho I_{LN} + \mathbb{I}^\intercal \mathbb{I}) = (L/\rho) + 1 \;,$$

and the inverse simplifies to

$$(\rho I_{LN} + \mathbb{I}^\intercal \mathbb{I})^{-1} = \frac{1}{\rho} I_{LN} - \frac{1}{\rho^2 + \rho L} \mathbb{I}\mathbb{I}^\intercal \;.$$

A. Multi-channel Signals

Multi-channel signals can be represented using either single channel dictionary filters and multi-channel coefficient maps or multi-channel dictionary filters and single channel coefficient maps [18]. Masked CSC for the former involves trivial modifications to the algorithms for single channel signals, and the masked CSC problem for the latter case can be expressed as

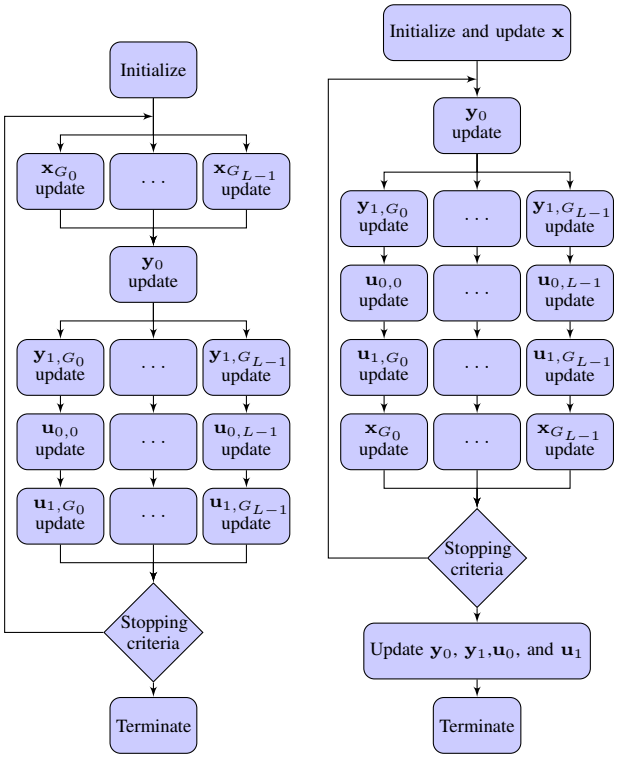
$$\underset{\{\mathbf{x}_m\}}{\operatorname{arg\,min}} \frac{1}{2} \sum_c \Big\| \mathbf{w} \odot \Big(\sum_m \mathbf{d}_{c,m} * \mathbf{x}_m - \mathbf{s}_c \Big) \Big\|_2^2 + \lambda \sum_m \| \mathbf{x}_m \|_1 \;,$$

where c indexes the C different channels. It is clear from inspection of this problem that steps (9) – (13) of the ADMM-DP algorithm are easily modified to solve this generalization of (8).

B. Complexity Comparison and Implementation

The computational complexities of setting up and solving each ADMM step for ADMM-EQ, ADMM-MD, and ADMM-DP are listed in Table I. The cost of solving the linear systems required to solve the $\mathbf{x}$ steps is linear in N, but setting up the systems requires DFTs which are computed in $\mathcal{O}(N \log N)$. Computationally, it is most efficient to include all other DFT operations in the $\mathbf{x}$ step as well. For the $\mathbf{x}$, $\mathbf{y}_1$, $\mathbf{u}_0$, and $\mathbf{u}_1$ steps for ADMM-DP, we report the complexity of each of the L disjoint problems that can be computed in parallel.

ADMM algorithms typically start each iteration with the update for the primary variable, $\mathbf{x}$, followed by those for the auxiliary variable, $\mathbf{y}$, and the dual variable, $\mathbf{u}$. Because our $\mathbf{y_0}$ update involves all groups, and cannot be computed in parallel, the direct implementation of the ADMM-DP algorithm breaks the parallelizable updates into two separate operations as shown in Fig. 1a. By cyclically permuting



(a) Direct Implementation (b) Efficient Implementation

Fig. 1: ADMM-DP implementation flowcharts.

TABLE II: Optimal Convergence Parameters

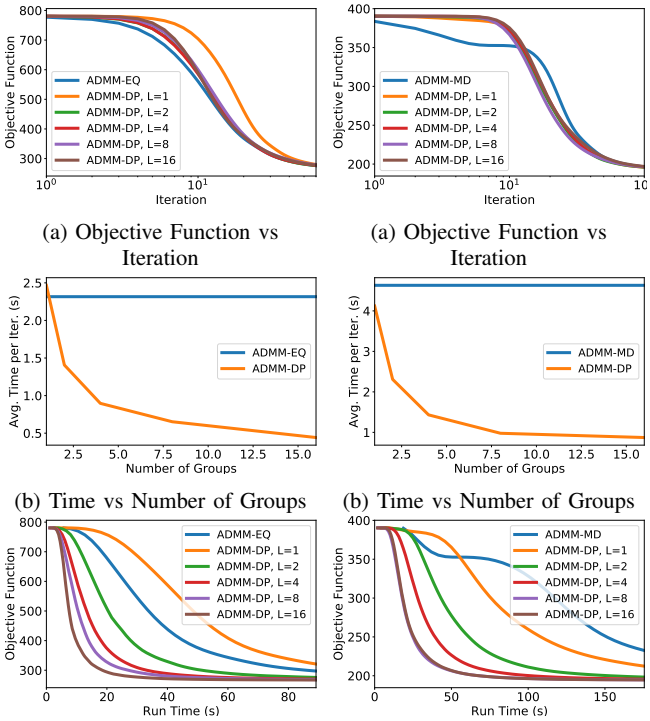
CSC Problem			Masked CSC Problem		
Method	ρ	α	Method	ρ	α
EQ	8.0	—	MD	5.0	—
DS L=1	6.0	1.1	DS L=1	1.0	2.4
DS L=2	2.0	2.0	DS L=2	1.0	2.4
DS L=4	2.0	2.0	DS L=4	1.0	2.4
DS L=8	2.0	2.0	DS L=8	2.0	1.8
DS L=16	2.0	2.0	DS L=16	2.0	1.8

the update steps, the parallel steps can be arranged consecutively, requiring only one merge per iteration as shown in Fig. 1b and implemented in practice.

IV. Results

All results were computed using a Linux workstation with 250GB of RAM, and with two Intel(R) Xeon(R) CPU E5-2640 v3 processors, providing a total of 16 physical cores and 32 logical cores. The dynamic frequency scaling feature of these CPUs, which was enabled for our experiments, reduces the effective speed improvement of the proposed approach. The ADMM-DP algorithm was implemented in Python, and compared with ADMM-EQ and ADMM-MD implementations in the SPORCO Python package [19].

The algorithms were compared for both problems (1) and (2) in sparse coding of a 512×512 pixel image with a dictionary of 192 filters of size 12×12, and with sparsity parameter $\lambda = 0.1$. The masked CSC problem used a mask of 0 and 1 entries in a checkerboard pattern. Optimal convergence parameters, shown in Table II, were determined by a grid search to minimize the objective values of each experiment after 250 iterations. For simplicity, experiments were performed without advanced techniques such as adaptive ρ or over-relaxation [12, Sec. III.D]. Convergence and run time comparisons



(a) Objective Function vs Iteration

(b) Time vs Number of Groups

(c) Objective Function vs Time

Fig. 2: Convergence and run times for CSC.

(a) Objective Function vs Iteration

(b) Time vs Number of Groups

(c) Objective Function vs Time

Fig. 3: Convergence and run times for masked CSC.

TABLE III: Iterations (denoted by 'Iter.") and time required to achieve specified accuracies in a masked CSC problem. Time was measured in seconds, and "Ratio" is the amount of time relative to ADMM-MD to reach the desired accuracy.

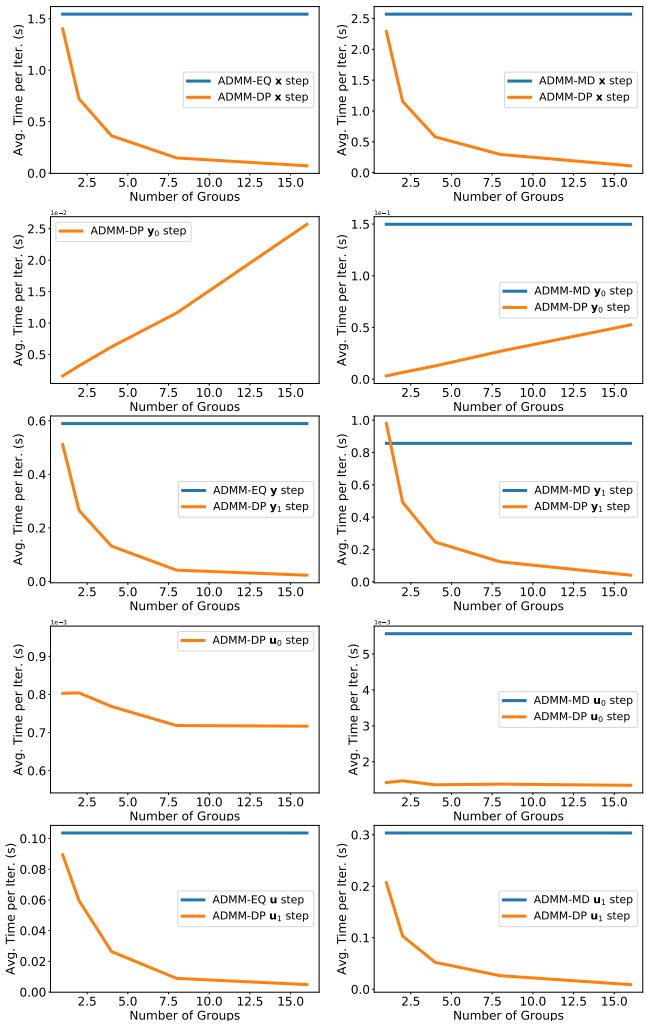
	Obj. Value error = 1% PSNR error $\approx 0.4\%$			Obj. Value error = 0.1% PSNR error $\approx 0.1\%$		
ADMM	Iter.	Time (s)	Ratio	Iter.	Time (s)	Ratio
MD	103	490.4	1.00	186	869.7	1.00
DS L=1	94	391.8	0.80	173	717.2	0.82
DS L=2	96	224.8	0.46	185	430.3	0.49
DS L=4	104	151.2	0.31	192	276.1	0.32
DS L=8	103	102.2	0.21	201	197.7	0.23
DS L=16	111	99.2	0.20	210	184.4	0.21

for the unmasked and masked CSC problems are shown in Figs. 2 and 3 respectively.

In order to analyze the computation time scaling with respect to L of the ADMM-DP algorithm, we computed the average time per iteration in solving our test problems for each of the algorithm steps, when executed in serial. The comparison with times spent on ADMM-EQ steps is shown in Fig. 4a, and the ADMM-MD comparison is in Fig. 4b. As an indication of the speed improvements that can be expected when the CSC problem is solved to an accuracy required in a real application, Table III reports the number of iterations, time in seconds, and time relative to ADMM-MD, to solve our masked CSC problem for the objective value to be within a percentage of the optimal objective value.

V. Discussion

There are two properties to consider when analyzing ADMM-DP: the convergence properties of the ADMM-DP methods relative to the standard methods, and the time scaling property from parallelization



(a) Standard CSC Times (b) Masked CSC Times

Fig. 4: ADMM-DP complexity measurements with respect to L in CSC problems.

of the ADMM-DP algorithm. Comparisons for the CSC problem without a spatial mask are shown in Fig. 2, and those for the masked problem are shown in Fig. 3.

It can be seen from Fig. 2a that the convergence of ADMM-EQ with respect to iterations is slightly better[2] than that of ADMM-DP. Run times per iteration are compared in Fig. 2b. Here ADMM-EQ and the ADMM-DP $L = 1$ are comparable, with ADMM-DP taking just $1.06\times$ longer. When $L = 16$, ADMM-DP was over $5.2\times$ faster per iteration relative to ADMM-EQ. Since the convergence for ADMM-EQ is slightly better than that of ADMM-DP, the effective speed-up is approximately $4.6\times$, as seen in Fig. 2c.

In Fig. 3a, ADMM-DP exhibits similar convergence to ADMM-MD when solving to high accuracy. With $L = 1$, ADMM-DP is $1.1\times$ faster per iteration than ADMM-MD, and with $L = 16$, ADMM-DP is $5.3\times$ faster than ADMM-MD. With similar convergence trends per iteration, the effective speed-up for an accurate solution is approximately $5\times$, as seen in Fig. 3c. This is confirmed by Table III, for which the masked CSC problem was solved to various accuracies measured by the objective value. Solving the problem with ADMM-DP, $L = 16$ to a 1% accuracy, or approximately a $.4\%$ PSNR error, is $5.0\times$ faster than ADMM-MD. To solve to an accuracy of 0.1%, or approximately a $.1\%$ PSNR error, is $4.7\times$ faster.

The time scaling of each step shown in Fig. 4 is in accordance with the complexities listed in Table I. It can be seen from Fig. 4 that the majority of the execution time of ADMM-EQ and ADMM-MD is spent on the $\mathbf{x}$ steps, which includes solving a linear system with complexity $\mathcal{O}(CMN \log N)$. ADMM-DP breaks this step up into L sub-problems each with complexity $\mathcal{O}(C\frac{M}{L}N \log N)$, resulting in the $\mathcal{O}(\frac{1}{L})$ trend observed in the ADMM-DP $\mathbf{x}$ steps. ADMM-DP similarly breaks up the $\mathbf{y}/\mathbf{y}_1$ and $\mathbf{u}/\mathbf{u}_1$ steps, giving a similar $\mathcal{O}(\frac{1}{L})$ trend here as well. As expected, the ADMM-DP $\mathbf{y}_0$ trend is linear, while the ADMM-DP $\mathbf{u}_0$ step takes approximately constant time.

The performance of ADMM-DP with larger L can also be predicted from Fig. 4. As L grows, the limited returns of the $\mathcal{O}(\frac{1}{L})$ terms will be dominated by the $\mathcal{O}(L)$ growth of ADMM-DP $\mathbf{y}_0$ step. It is clear from Figs. 2b and 3b that our experiments did not reach that transition point, and would still benefit from more cores. The trends observed in Fig. 4 suggest, however, that using many more groups, such as would be possible with the very large number of cores available on a GPU, would not be efficient. We expect that the most effective approach on such an architecture would be to partition the available cores into 16 to 32 sets, each of which would be assigned a single dictionary filter group.

VI. Conclusion

We have proposed a new parallelizable CSC algorithm that offers significant performance gains on multi-core architectures, which are becoming increasingly prevalent. GPU implementations of existing algorithms can offer considerably greater time reductions, but are limited by hardware availability, and are much more time-consuming to program and modify.

Some of our tests indicate that the use of the Python multiprocessing module introduces a significant overhead, reducing the effective gain obtained from computing in parallel. We expect that an implementation using alternative parallelization tools may eliminate this expense, further improving the speed improvement.

Implementations of the algorithms proposed here will be included in a future release of the SPORCO library [19], [20].

[2]It is substantially better when ADMM-DP does not make use of the constraint-balancing offered by parameter α, i.e. when $\alpha = 1$.

References

[1] J. Mairal, F. Bach, and J. Ponce, "Sparse modeling for image and vision processing," *Foundations and Trends in Computer Graphics and Vision*, vol. 8, no. 2-3, pp. 85–283, 2014. doi:10.1561/0600000058

[2] S. Gu, W. Zuo, Q. Xie, D. Meng, X. Feng, and L. Zhang, "Convolutional sparse coding for image super-resolution," in *Proc. IEEE Intl. Conf. Comput. Vis. (ICCV)*, Dec. 2015. doi:10.1109/ICCV.2015.212

[3] Y. Liu, X. Chen, R. K. Ward, and Z. J. Wang, "Image fusion with convolutional sparse representation," *IEEE Signal Process. Lett.*, 2016. doi:10.1109/lsp.2016.2618776

[4] H. Zhang and V. Patel, "Convolutional sparse coding-based image decomposition," in *British Mach. Vis. Conf. (BMVC)*, York, UK, Sep. 2016.

[5] T. M. Quan and W.-K. Jeong, "Compressed sensing reconstruction of dynamic contrast enhanced MRI using GPU-accelerated convolutional sparse coding," in *IEEE Intl. Symp. Biomed. Imag. (ISBI)*, Apr. 2016, pp. 518–521. doi:10.1109/ISBI.2016.7493321

[6] B. Wohlberg, "Convolutional sparse representations as an image model for impulse noise restoration," in *Proc. IEEE Image, Video Multidim. Signal Process. Workshop (IVMSP)*, Bordeaux, France, Jul. 2016. doi:10.1109/IVMSPW.2016.7528229

[7] A. Serrano, F. Heide, D. Gutierrez, G. Wetzstein, and B. Masia, "Convolutional sparse coding for high dynamic range imaging," *Computer Graphics Forum*, vol. 35, no. 2, pp. 153–163, May 2016. doi:10.1111/cgf.12819

[8] H. Zhang and V. M. Patel, "Convolutional sparse and low-rank coding-based rain streak removal," in *Proc. IEEE Winter Conference on Applications of Computer Vision (WACV)*, March 2017.

[9] R. Chalasani, J. C. Principe, and N. Ramakrishnan, "A fast proximal method for convolutional sparse coding," in *Proc. Int. Joint Conf. Neural Net. (IJCNN)*, Aug. 2013.

[10] H. Bristow, A. Eriksson, and S. Lucey, "Fast convolutional sparse coding," in *Proc. IEEE Conf. Comp. Vis. Pat. Recog. (CVPR)*, Jun. 2013, pp. 391–398. doi:10.1109/CVPR.2013.57

[11] B. Wohlberg, "Efficient convolutional sparse coding," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, May 2014, pp. 7173–7177. doi:10.1109/ICASSP.2014.6854992

[12] ——, "Efficient algorithms for convolutional sparse representations," *IEEE Trans. Image Process.*, vol. 25, no. 1, pp. 301–315, Jan. 2016. doi:10.1109/TIP.2015.2495260

[13] F. Heide, W. Heidrich, and G. Wetzstein, "Fast and flexible convolutional sparse coding," in *Proc. IEEE Conf. Comp. Vis. Pat. Recog. (CVPR)*, Jun. 2015, pp. 5135–5143. doi:10.1109/CVPR.2015.7299149

[14] B. Wohlberg, "Boundary handling for convolutional sparse representations," in *Proc. IEEE Conf. Image Process. (ICIP)*, Phoenix, AZ, USA, Sep. 2016, pp. 1833–1837. doi:10.1109/ICIP.2016.7532675

[15] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2010. doi:10.1561/2200000016

[16] M. S. C. Almeida and M. A. T. Figueiredo, "Deconvolving images with unknown boundaries using the alternating direction method of multipliers," *IEEE Trans. Image Process.*, vol. 22, no. 8, pp. 3074–3086, Aug. 2013. doi:10.1109/tip.2013.2258354

[17] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 127–239, 2014. doi:10.1561/2400000003

[18] B. Wohlberg, "Convolutional sparse representation of color images," in *Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI)*, Santa Fe, NM, USA, Mar. 2016, pp. 57–60. doi:10.1109/SSIAI.2016.7459174

[19] ——, "SPORCO: A Python package for standard and convolutional sparse representations," in *Proceedings of the 15th Python in Science Conference*, Austin, TX, USA, Jul. 2017, pp. 1–8. doi:10.25080/shinma-7f4c6e7-001

[20] ——, "SParse Optimization Research COde (SPORCO)," Software library available from http://purl.org/brendt/software/sporco, 2017.