Politehnica University of Bucharest
Faculty of Automatic Control and Computers
Department of Automatic Control and Systems Engineering

PHD THESIS

Parallel Dictionary Learning Algorithms for Sparse Representations

Paul Irofti

Advisor
Prof. dr. ing. Bogdan Dumitrescu

Bucharest, 2015

Contents

Chapter 1

Prologue

Sparse representations are intensively used in signal processing applications, like image coding, denoising, echo channels modeling, compression and many others. Recent research has shown encouraging results when the sparse signals are created through the use of a learned dictionary. This raised the following optimization problem

$$\begin{aligned} \underset{D,X}{\text{minimize}} \quad & \|Y - DX\|_F^2 \\ \text{subject to} \quad & \|x\|_0 \leq s, \ \forall x \in X, \end{aligned}$$

that attempts to find the best dictionary D generating the best sparse representations X when modeling a given set of signals Y with a target sparsity of s imposed on each sparse signal x from X. This is a hard open problem for which the signal processing field has been able, so far, to only offer local minima solutions that are usually very involved and take a long time to process.

The current study focuses on finding new methods and algorithms, that have a parallel form where possible, for obtaining sparse representations of signals with improved dictionaries that lead to better performance in both representation error and execution time.

In Part I we start by describing in Chapter 2 the tools for obtaining the sparsest solutions and the guarantees they provide along with the currently available dictionary design strategies. We continue with Chapter 3 where we present the OpenCL parallelism framework along with the analysis and efficiency indicators for optimal implementations.

In Part II we attack the general dictionary learning problem by first investigating and proposing new solutions for the sparse representation stage in Chapter 4 and then moving on to the dictionary update stage in Chapter 5 where we propose a new parallel update strategy and describe its effect

on existing algorithms. Lastly, we study in Chapter 6 how mixing different representation algorithms with different dictionary update methods affects the quality of the final dictionary.

Part III focuses on dictionary learning solutions where the dictionary has a specific form. In Chapter 7 we propose a new parallel algorithm for dictionaries structured as a union of orthonormal bases. Next, in Chapter 8, we study the cosparse view on dictionary learning and propose new algorithms for creating cosparse orthogonal dictionaries. Finally, in Chapter 9 we analyse denosing through dictionary learning and propose new methods based on composite dictionaries.

We conclude this thesis in Part IV where we list and describe our contributions and present future research directions.

Chapter 2

Dictionary Learning

We start this chapter by presenting the sparse representation field [1] and its guarantees in providing the sparsest solution to a linear system of equations. In the second part we shift toward the design strategies of the representation matrix.

2.1 The l_0 Pseudo-Norm

We attempt to provide sparse solutions for the underdetermined linear systems of equations of the form

$$Ax = b, \tag{2.1}$$

where $A \in \mathbb{R}^{n \times m}$ is full rank, with $n < m$.

Our interest is providing the sparsest solutions possible and so we will focus almost exclusively on the l_0 pseudo-norm which acts as a non-zero indicator on the support of a given vector x:

$$\|x\|_0 = |\{i : x_i \neq 0\}|, \tag{2.2}$$

where $|.|$ denotes the cardinality of the set.

The l_0 optimization problem for solving (2.1) is

$$\min_x \|x\|_0 \quad \text{subject to} \quad Ax = b. \tag{2.3}$$

New and existing solutions, algorithms and implementations for this optimization problem are discussed and analyzed in detail in Chapter 4.

2.2 When is a solution the sparsest solution?

Knowing that l_0 provides us with the most sparse solution possible we present here some of its known properties such as uniqueness, stability and performance in order to motivate its core position in our study.

2.2.1 Uniqueness Constructs

Given a sparse solution x of the linear system $Ax = b$, can we tell if it is the sparsest one? In this subsection we present the tools available for answering this question.

The first result is based on the property of the matrix A called the *spark* [2], which represents the smallest number of linearly-dependent columns of matrix A. The spark is used in the following theorem:

Theorem 2.1 *If a given sparse solution x of the linear system $Ax = b$ has $\|x\|_0 < spark(A)/2$ then it is also the sparsest.*

Finding the spark is not an easy task because it involves sweeping through all possible column subsets and perhaps that is why weaker but faster bounds such as the mutual-coherence [3] bound are preferred. The mutual-coherence of a matrix A ($\mu(A)$) is the absolute largest normalized inner-product of its columns. This is used to guarantee that the sparsest solution was reached [4]:

Theorem 2.2 *If a given sparse solution x of the linear system $Ax = b$ has $\|x\|_0 < \frac{1}{2}\left(1 + \frac{1}{\mu(A)}\right)$ then it is also the sparsest.*

2.2.2 Stability for Sparse Approximations

When performing sparse approximations, instead of exact solutions, the uniqueness of a given solution, using tools like the ones presented in the last subsection, can no longer be claimed.

Approximations can present feasible solutions that are susceptible to scaling and, worse, they can have different supports while maintaining the same l_0 cardinality that still make them the sparsest possible. We are interested in the stability [5] of these solutions: if a solution is found that is also the sparsest we want to be sure that it is also the best within a given margin ϵ.

A mutual-coherence based result from [5] states that:

Theorem 2.3 *Given a sparse solution x_0 that satisfies the mutual-coherence criteria from Theorem 2.2 and that provides an approximation within a given*

error ϵ ($\|b - Ax_0\| < \epsilon$), then the distance to every other feasible sparsest solution x is bounded by

$$\|x - x_0\|_2^2 \leq \frac{4\epsilon^2}{1 - \mu(A)(2\|x_0\|_0 - 1)}. \tag{2.4}$$

Another powerful tool that can also be used for stability analysis is the restricted isometry property [6] (RIP):

Definition 2.1 *Given a matrix A with normalized columns and an integer s such that A_s represents the matrix A restricted to only s columns, suppose there exists a quantity δ_s, that is also the smallest, such that for all possible submatrices A_s the following holds true:*

$$(1 - \delta_s)\|c\|_2^2 \leq \|A_s c\|_2^2 \leq (1 + \delta_s)\|c\|_2^2 \quad \forall c \in \mathbb{R}^s. \tag{2.5}$$

Then A is said to have an s-RIP with a δ_s constant.

Using this definition it has been shown in [1] that we can get to the same stability claims from Theorem 2.3.

2.3 Dictionary Types

In this section we turn our focus towards the properties of A and its sparsifying effect on the final solution x. From this perspective the literature often refers to matrix A as a representation dictionary D whose columns are termed atoms.

The crude dictionary form is built from popular matrix transforms such as Fourier, discrete cosine or wavelets whose components are vectorized as dictionary atoms. The created dictionary remains fixed and is used universally for the representation of all signal types.

A different approach uses pre-designed, but still fixed, dictionaries that are used only for specific classes of signals. Such tuned dictionaries can be built using something like wavelet packets, bandlets, curvelets or contourlet. While this loses the generality of the former universal approach it provides substantially better representations. Even so, when we are provided with such a dictionary, we can still obtain mediocre results on our signal sets due to D being too specialized or too general.

2.4 Learning

The shortcomings of the above methods have paved the way for a third versatile approach called dictionary learning (DL) [7, 8]. The central idea in

Algorithm 1: Dictionary learning – general structure

```
Arguments: signal matrix Y, target sparsity s
Initialize: dictionary D (with normalized atoms)
for k = 1, 2, ... do
    With fixed D, compute sparse representations X
    With fixed X, update atoms d_j, j = 1 : n
```

DL is to first train a specific dictionary on a relevant training set until a well-suited specialized final dictionary is obtained that can be used in fixed form to perform efficient representations of other signal sets of the same class. This strategy has been shown [9] to be very well suited for sparse representations and our study will gravitate around this approach when investigating and proposing new dictionary designs methods.

Formally, the DL problem is posed as follows. Given a data set $Y \in \mathbb{R}^{p\times m}$, made of m vectors (signals or data items) of size p, and a sparsity level s, the aim is to solve the optimization problem

$$\begin{aligned} \underset{D,X}{\text{minimize}} \quad & \|Y - DX\|_F^2 \\ \text{subject to} \quad & \|x_i\|_0 \leq s,\ 1 \leq i \leq m, \end{aligned} \tag{2.6}$$

where the variables are the atoms of dictionary $D \in \mathbb{R}^{p\times n}$, and the sparse representations matrix $X \in \mathbb{R}^{n\times m}$, whose columns have at most s nonzero elements. By x_i we denote the i-th column of the matrix X and by $\|\cdot\|_F$ the Frobenius norm of a matrix. In practice, the dictionary D is initialized either randomly or by a random selection of signal vectors. The norm of the atoms is forced to 1 at all stages, in order to eliminate the multiplicative indeterminacy in the product DX.

Algorithm 1 presents the general structure of most DL algorithms. First, a sparse representation algorithm is used to compute the sparse representation matrix X. Then, the atoms are updated using different methods all aiming to reduce the objective of (2.6).

Among the existing dictionary design algorithms that attempt to solve problem (2.6) we mention method of optimal directions (MOD) [10], K-SVD [11], approximate K-SVD (AK-SVD) [12], sequential generalization of K-means (SGK) [13], new SGK (NSGK) [14], union of orthonormal basis(UONB) [15], and single block orthogonal algorithm (SBO) [16] which we discuss in Chapters 5 and 7. Our research proposes parallel alternatives for most of the above algorithms and provides a new parallel DL framework called Jacobi Atom Updates (JAU).

Chapter 3

Parallelism with OpenCL

OpenCL is an open standard allowing portable parallel programming, aimed especially at graphics processing units (GPU) but not restrained to them. Despite its recent proposal, OpenCL has gained support from the industry and its implementation is supported by the major GPU manufacturers. Although some implementations miss certain features and there are difficulties in portability, there are much more incentives for using OpenCL than languages specialized to a single type of GPUs.

Hardware Abstraction OpenCL abstracts the smallest execution unit available in hardware as processing elements (PE), that are organized in equally sized groups at which parallelism is guaranteed, called compute units (CU), located on the OpenCL device. Compute units can also be executed in parallel. Each PE has its own private memory, which is the fastest type of memory on the OpenCL device but also the smallest. PEs share resources locally, within the compute unit, and globally, on the OpenCL device. The local memory of a CU is visible only to its PEs. The global memory is the only type of memory accessible from the outside.

Scheduling Work The standard allows a task to request a number of individual processing items called work-items that are all guaranteed to be executed on the device's PEs, but the parallelism is left up to the scheduler. The task can group work-items into work-groups. The work-items within one work-group are executed by the PEs from a single compute unit.

Topology Work-items consist of identical small functions (also called kernels) and are organized in an n-dimensional space that is defined in software by the application. This logical split allows to differentiate work among work-items through indexing. The chosen dimension creates a tuple of cartesian

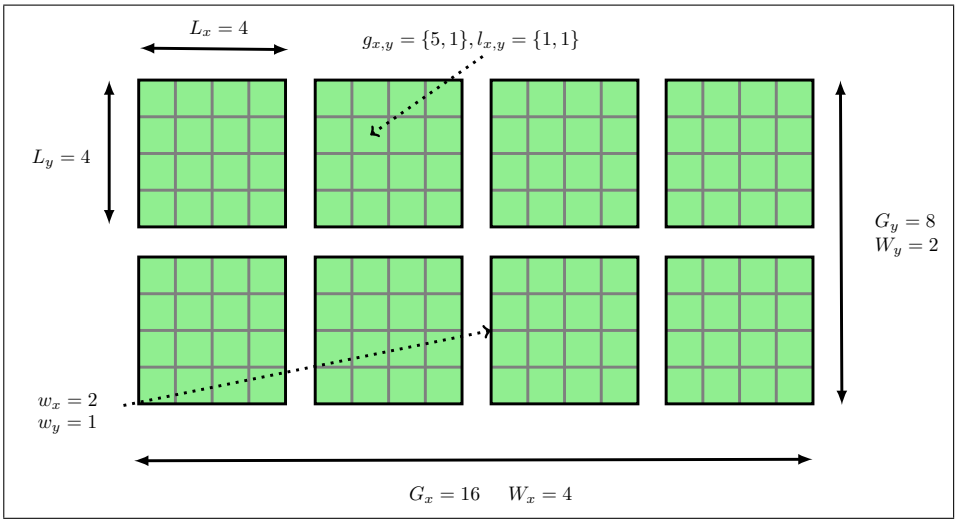

Figure 3.1: 2-dimensional split example where indexing starts from 0

coordinates that define global and local IDs. For a bidimensional split of the PE set, we can denote the n-dimensional range definition as NDR($\langle G_x, G_y \rangle$, $\langle L_x, L_y \rangle$). There are $G_x \times G_y$ PEs, organized in work-groups of size $L_x \times L_y$, running the same kernel.

GPU Particularities While GPUs are able to successfully model the OpenCL abstract device the hardware underneath is usually quite different. GPUs have a much smaller memory when compared to CPUs and their processing elements operate at lower frequencies. Private memory is represented by GPU register memory which is built from the set of vector general purpose registers (VGPRs). A compute unit on the GPU is composed of one or more wavefronts. Wavefront is the smallest grouping at which parallelism takes place. The PEs within have lock-step instruction execution.

Performance and Occupancy The main goal when designing an OpenCL kernel is performance through parallelism. When the target is a GPU device, this often implies ensuring full resource occupancy. In other words we need to make sure that all the PEs are actively processing at any given time during the execution of our kernel.

There are of course corner cases where the best performance does not imply full occupancy like when the time cost of transferring data from the host to the OpenCL device or from global to local memory is higher than the actual execution time.

Chapter 4

Sparse Representations

4.1 Introduction

When looking at the sparse representation problem we are interested in finding the representation with the largest number of zeros in its support that uses a known fixed dictionary to represent a given full signal. This can be formalized as the following optimization problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \|x\|_0 \\ \text{subject to} \quad & y = Dx, \end{aligned} \tag{4.1}$$

where y is the signal, D the dictionary, and x the resulting sparse representation. This is a hard problem and most of the existing methods propose an alternative to (4.1) by approximating y following a sparsity constraint s:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \|y - Dx\|_2^2 \\ \text{subject to} \quad & \|x\|_0 \leq s. \end{aligned} \tag{4.2}$$

We can split the pursuit of x in two parts: finding the best few columns from D, to be used as the support of x, and then filling its non-zero entries with the coefficients found through least-squares (LS). Denoting $\mathcal{I}$ as the support set and $D_{\mathcal{I}}$ the restriction of D to the columns belonging to $\mathcal{I}$, we can compute the representation as $x_{\mathcal{I}} = (D_{\mathcal{I}}^T D_{\mathcal{I}})^{-1} D_{\mathcal{I}}^T y$, where $x_{\mathcal{I}}$ are the coefficients corresponding to the current support and thus the other elements of x are zero.

The popular dictionary design algorithms that we will present in Chapter 5 (MOD [10], K-SVD [11], AK-SVD [12], SGK [13], NSGK [14]) use Orthogonal Matching Pursuit (OMP) [17] in the first stage to compute the sparse representations. The main reason is that OMP is fast and also that it is used

Algorithm 2: Projection-Based Orthogonal Least Squares (POLS)

Arguments: A, b, s, L
Initialize: $\mathcal{I} = \emptyset$
for $k = 1 : s$ **do**
    for $j \notin \mathcal{I}$ **do**
        Build new support: $\mathcal{J} = \mathcal{I} \cup \{j\}$
        Try solution: $x = \mathrm{LS}(A, b, \mathcal{J})$
        Residual norm: $\rho_j = \|b - A_{\mathcal{J}} x_{\mathcal{J}}\|^2$
    Select indices $\mathcal{J}$ of L largest ρ_j
    Compute potential solution: $x = \mathrm{LS}(A, b, \mathcal{I} \cup \mathcal{J})$
    Select largest element index: $i = \arg\max_{j \in \mathcal{J}} |x_j|$
    Increase support: $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$
    Compute new solution: $x = \mathrm{LS}(A, b, \mathcal{I})$

in applications together with the optimized dictionary; it makes sense to appeal to the same representation algorithm in training the dictionary as well as in using it. However, there are other greedy algorithms, like Orthogonal Least Squares (OLS) [18], Projection-Based OMP (POMP) or Look-Ahead OLS (LAOLS) [19], that are still fast enough for wide practical use, but achieving typically better representations than OMP.

Our first contribution here is a new sparse representation algorithm POLS. The second contribution is orientated towards the technical details of efficiently implementing the representation algorithms for heterogeneous parallel systems.

4.2 Greedy Minimizations

A refinement of OMP was proposed in [19], in the form of Projection-Based OMP (POMP). Unlike OMP, a number of L candidate columns are selected via the matching pursuit criterion, where L is an argument of the algorithm. Then, an LS solution is computed for the support extended with all these columns. The winner is the column with the largest element of the solution.

An immediate extension, not investigated until now, is the Projection-Based OLS (POLS), presented in Algorithm 2, which is the application of the POMP selection idea in the context of OLS.

Algorithm complexity. POMP performs an extra LS operation on the support and its L-sized extension $\mathcal{J}$. This amounts to L incremental LS calculations at each iteration, which results in a total $\mathcal{O}(sL(s+L)p)$ extra

cost compared to plain OMP. If $L = \mathcal{O}(s)$, then the extra cost is $\mathcal{O}(s^3 p)$. POLS adds a single computationally significant instruction to OLS which is, again, the LS on the extended support.

The increase in complexity for POMP and POLS with respect to OMP and OLS is negligible for small s, but becomes significant when $s^2 > n$. (We assume that $L \leq s$, which is a good practical choice.)

4.3 Parallelism

The dictionary learning process, as described in Chapter 2, operates on large training signals sets that need to be sparsely represented. And so, following the equation from (4.2), the first stage of the dictionary learning process is naturally parallel, since the signal representations are completely independent. Given that this applies to all algorithms described in Section 4.2, we picked Batch OMP(BOMP) [12] as a case-study as it is the popular choice in the literature. In the following subsections we will present the details of the OMP algorithm and describe its parallel OpenCL implementation.

4.3.1 Batch Orthogonal Matching Pursuit

The matrix precomputations needed by BOMP were performed by a dedicated BLAS kernel that implements block matrix multiplication. Since the two multiplications are independent of each other, they can be also performed in parallel. All the operations required for the sparse representation of a single signal, were packed in and implemented by a single OpenCL kernel.

The main obstacles we encountered during the implementation were memory bound. BOMP is a huge memory consumer and mostly due to auxiliary data. The necessary memory is of size $O(ns)$. Keeping all the auxiliary data in local memory would permit only the processing of one signal per compute-unit, corresponding to an $NDR(\langle \tilde{m} \rangle, \langle 1 \rangle)$ splitting, where $\tilde{m} \leq m$. This would be wasteful as it would not reach full GPU occupancy and thus it would not cover the global memory latency costs. Figure 4.1 shows how resources limit the number of active wavefronts (our kernel is marked with a square dot).

4.3.2 Performance

In this subsection we present the performance of the parallel GPU implementation of the BOMP algorithm and compare it to an almost identical CPU version. We were able to keep an almost one-to-one instruction equivalence

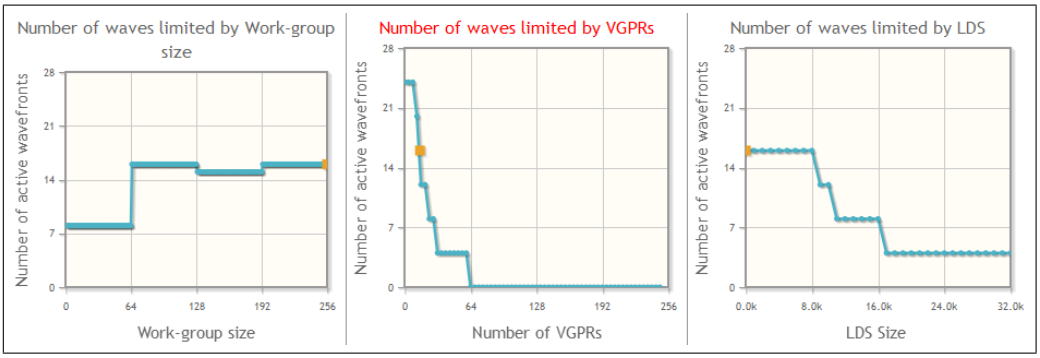

Figure 4.1: BOMP representation kernel occupancy

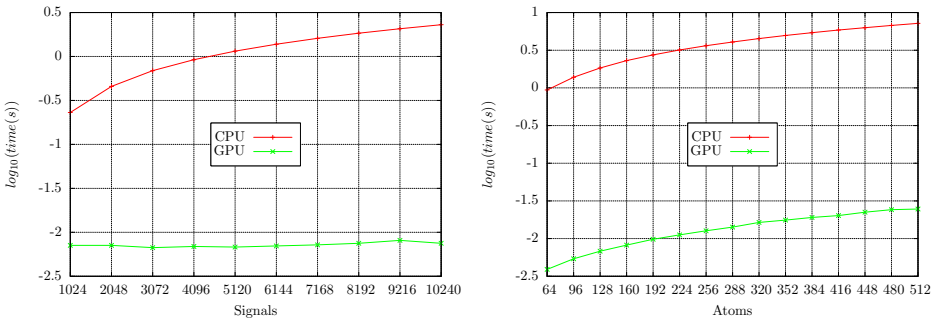

Figure 4.2: BOMP performance as we increase the of signals (left) and atoms (right).

due to the fact that the OpenCL langauge is a custom subset of the C language. We measured the execution times when varying the number of signals and keeping a fixed dictionary dimension and vice-versa. In both scenarios we used $\tilde{m} = m$ in the OpenCL implementation.

The left part from Figure 4.2 presents the elapsed time when representing a signals set with a varied size between $m = 1024$ to $m = 10240$ with a fixed dictionary of $n = 128$ atoms of $p = 64$ size each and a sparsity goal of $s = 8$. This experiment shows a performance improvement of up to 312 times when using the OpenCL GPU version.

On the right, the performance improvement of the GPU implementation is clearly visible as increasing the dictionary size ($n = 64$ up to $n = 512$) has a direct impact on the number of instructions performed by each work-item. In our tests we used a fixed number of $m = 8192$ signals of dimension $p = 64$ and a target sparsity of $s = 8$. Here, the GPU version is up to 250 times faster.

Chapter 5

Dictionary Design

Given a set of signals $Y \in \mathbb{R}^{p \times m}$ and a sparsity level s, the goal is to find a dictionary $D \in \mathbb{R}^{p \times n}$ that minimizes the Frobenius norm of the approximation error

$$E = Y - DX, \tag{5.1}$$

where $X \in \mathbb{R}^{n \times m}$ is the associated s-sparse representations matrix, with at most s nonzero elements on each column. This is the dictionary learning problem from Equation (2.6) with known sparse representations.

5.1 Atom update methods

The general structure of most DL algorithms was presented and described around Algorithm 1 from Chapter 2. The iterative process is composed of the two main stages depicted by steps 4 and 5 in the algorithm. In this section we will present several techniques for updating atom d_j in step 5. We denote $\mathcal{I}_j$ the set of the indices of nonzero elements on the j-th row of X. Otherwise said, these are the indices of the signals whose representation involves the atom d_j.

K-SVD [11] solves the optimization problem

$$\min_{d_j, X_{j,\mathcal{I}_j}} \left\| \left(Y_{\mathcal{J}} - \sum_{\ell \neq j} d_\ell X_{\ell,\mathcal{I}_\ell} \right) - d_j X_{j,\mathcal{I}_j} \right\|_F^2, \tag{5.2}$$

where all atoms excepting d_j are fixed. Note that the matrix within parenthesis is the representation error of the dictionary without the atom d_j. To minimize the error, the problem (5.2) is seen as a rank-1 approximation of this modified error matrix. The solution is given by the singular vectors corresponding to the largest singular value. The less complex AK-SVD [12]

runs a single iteration of the power method to compute the two vectors. Note that (5.2) allows the representations to be changed also in this stage.

SGK [13] considers the same optimization problem, but only with d_j as variable:

$$\min_{d_j} \left\| \left(Y_{\mathcal{J}} - \sum_{\ell \neq j} d_\ell X_{\ell,\mathcal{I}_\ell} \right) - d_j X_{j,\mathcal{I}_j} \right\|_F^2 . \tag{5.3}$$

This is a simple least-squares problem, for which an explicit solution can be easily computed. NSGK [14] operates similarly, but using differences with the previous values of the dictionary and representations.

5.2 Jacobi Atom Updates Stategy

All the algorithms from Section 5.1 update the atoms sequentially, using thus their most recent values when updating another atom. This is the typical Gauss-Seidel approach. We investigate here the Jacobi version, in which the atoms are updated independently, meaning that problems like (5.2) or (5.3) are solved simultaneously for $j = 1 : n$. Jacobi atom updates (JAU) can be applied to all presented dictionary update algorithms. We append the initial P (from parallel) to their name to denote the Jacobi versions.

The general form of the proposed dictionary learning method with Jacobi atom updates is presented in Algorithm 3. At iteration k of the DL method, the two usual stages are performed. In step 1, the current dictionary $D^{(k)}$ and the signals Y are used to find the sparse representation matrix $X^{(k)}$ with s nonzero elements on each column; we used OMP, as widely done in the literature.

The atom update stage takes place in groups of $\tilde{n}$ atoms. We assume that $\tilde{n}$ divides n only for the simplicity of description, but this is not a mandatory condition. Steps 2 and 3 of Algorithm 3 perform a full sweep of the atoms. All the $\tilde{n}$ atoms from the same group are updated independently (step 4), using one of the various available rules; some of them will be discussed in the next section. Once a group is processed, its updated atoms are used for updating the other atoms; so, atom $d_j^{(k+1)}$ (column j of $D^{(k+1)}$) is computed in step 4 using $d_i^{(k+1)}$ if

$$\lfloor (i-1)/\tilde{n} \rfloor < \lfloor (j-1)/\tilde{n} \rfloor, \tag{5.4}$$

i.e. $i < j$ and d_i not in the same group as d_j, and $d_i^{(k)}$ otherwise.

Putting $\tilde{n} = 1$ gives the usual sequential Gauss-Seidel form. Taking $\tilde{n} = n$ leads to a fully parallel update, i.e. the form that is typically labeled with Jacobi's name.

Algorithm 3: General structure of a DL-JAU iteration

Data: current dictionary $D^{(k)} \in \mathbb{R}^{p\times n}$
signals set $Y \in \mathbb{R}^{p\times m}$
number of parallel atoms $\tilde{n}$
Result: next dictionary $D^{(k+1)}$

1 Compute s-sparse representations $X^{(k)} \in \mathbb{R}^{n\times m}$ such that $Y \approx D^{(k)}X^{(k)}$
2 **for** $\ell = 1$ **to** $n/\tilde{n}$ **do**
3 **for** $j = (\ell-1)\tilde{n}+1$ **to** $\ell\tilde{n}$, *in parallel* **do**
4 Compute $d_j^{(k+1)}$
5 Update $d_j^{(k+1)} \leftarrow \alpha d_j^{(k+1)} + (1-\alpha)d_j^{(k)}$
6 Normalize: $d_j^{(k+1)} \leftarrow d_j^{(k+1)}/\|d_j^{(k+1)}\|$

We also propose (step 5) to update an atom via a convex combination of its new and former value, with a weight $\alpha \in (0, 1]$. The choice $\alpha = 1$ was always used until now, which is meaningful for the sequential approach, where we seek the optimal value of an atom, given all the others. In a parallel context, where a group of atoms are optimized simultaneously, it may be wise to temper their progress, since their independent evolution may overreach the target. Finally, step 6 is the usual normalization constraint on the dictionary.

The proposed form has obvious potential for a smaller execution time on a parallel architecture. We touch this issue in Section 5.3 where we present a case-study of a GPU implementation of the AK-SVD algorithm (also published in [20]). Here, we will focus solely on the quality of the designed dictionary.

5.2.1 Numerical Results

We give here numerical evidence supporting the advantages of the JAU scheme, for dictionary recovery and sparse image representation. We compare the JAU algorithms PAK-SVD, P-SGK and P-NSGK with their sequential counterparts. We report results obtained with the same input data for all the algorithms; in particular, the initial dictionary is the same. The sparse representations were computed via OMP[1].

[1]We used OMP-Box version 10 available at `http://www.cs.technion.ac.il/~ronrubin/software.html`

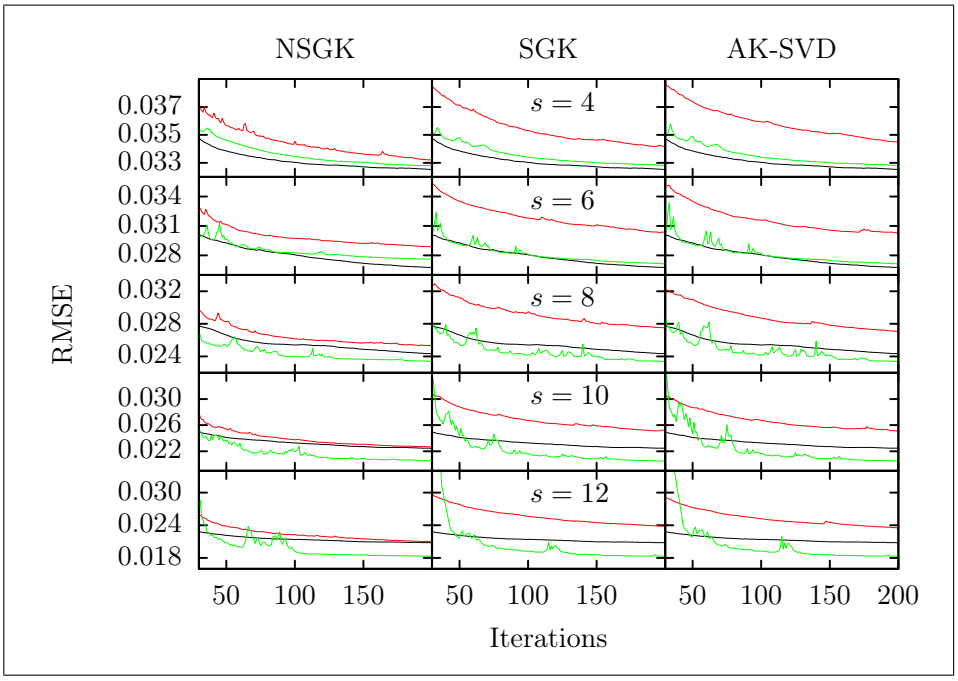

Figure 5.1: Error evolution at different sparsity constraints. The sequential versions are red, JAU algorithms green and MOD is black.

5.2.2 JAU versus MOD

We compare the performance in representation error of the JAU algorithms with the intrinsically parallel algorithm named method of optimal directions (MOD) [10]. MOD uses OMP for representation and updates the dictionary D with the least-squares solution of the linear system $DX = Y$. For completeness we also include the sequential versions on which JAU algorithms are built.

We present in Figure 5.1 the error improvement at each iteration for all algorithms, for several sparsity levels. In this experiment we used a dictionary of $n = 128$ atoms and a training set of $m = 8192$ signals. We can see that the JAU versions can jump back and forwards, specially during the first iterations. We think that this is due to the parallel update of the dictionary atoms which leads to jumps from one local minima to another until a stable point is reached. This is, perhaps, the reason why in the end it manages to provide a lower representation error. Even though the JAU convergence is not as smooth as MOD or the sequential versions, it has a consistent descendent trend.

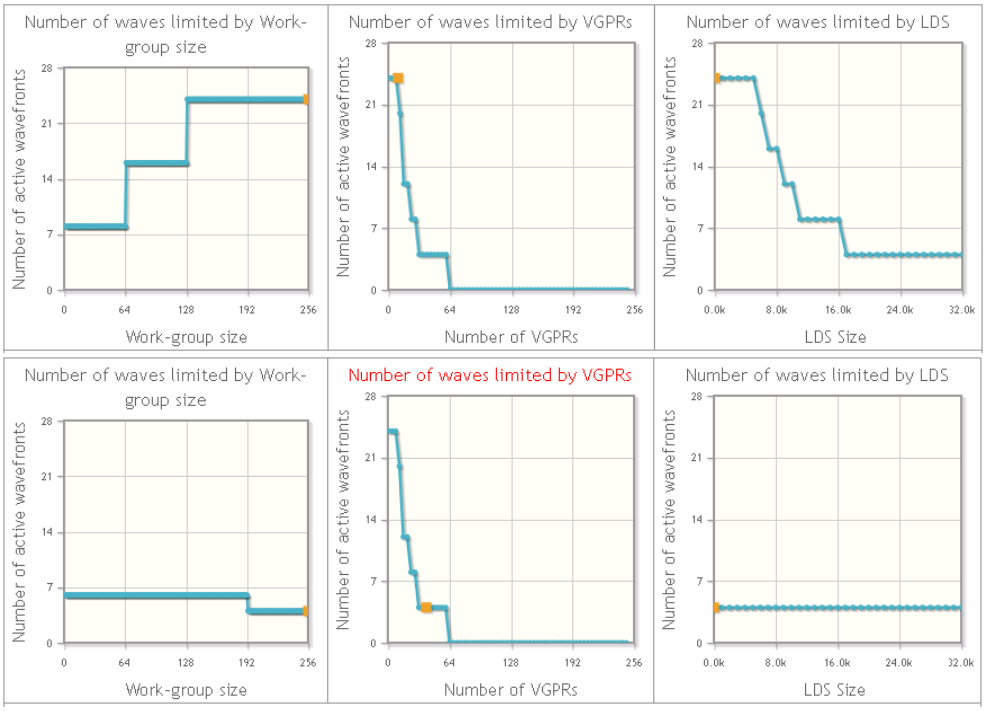

Figure 5.2: PAK-SVD dictionary update GPU occupancy. The top panel shows the benefits of keeping $\mathcal{I}$ in local memory while the bottom panel depicts what happens when we move it in global memory.

5.3 Parallel Dictionary Learning with OpenCL

In this section we focus on the study of efficient OpenCL implementations for the dictionary update stages of the AK-SVD, SGK and NSGK methods. This, together with the parallel results for the sparse representation stage described in Chapter 4, allows us to perform the entire dictionary learning process in parallel on the GPU.

Occupancy. In Figure 5.2 we analyze the atom update kernel performance in terms of GPU occupancy. Our experiments showed that varying the number of atoms in the dictionary from $n = 64$ up to $n = 512$ had no effect on occupancy. That is why we focus here on the main obstacle: the memory storage location of the indices set $\mathcal{I}$. We have shown that moving the indices in local memory has a significant impact bumping occupancy from 16% to 100% due to lowering the number of used VGPRs by 32 and 30 in the P-SGK and, respectively, PAK-SVD case. The difference can also be spotted by comparing the VGPR screens from the top and bottom panels. The rest of the occupancy factors are not affected by $\mathcal{I}$.

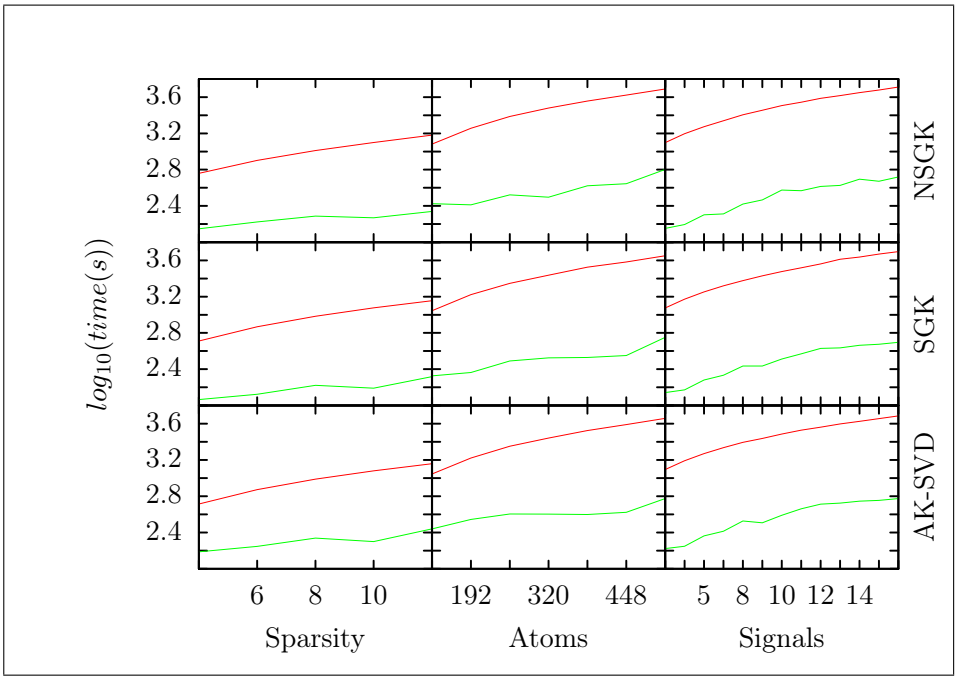

Figure 5.3: Execution times. The sequential versions are red and the JAU algorithms are green.

5.3.1 Sparsity, Atoms and Training Set Influence

Here we analyze the execution time improvements brought by our JAU strategy when applied on AK-SVD, SGK and NSGK by varying one dictionary design parameter (such as the sparsity constraint) and keep the others fixed (the number of dictionary atoms and training signals).

We present 3 experiments in Figure 5.3 where we vary the sparsity constraint, the atoms in the dictionary and the number of signals in the training set. We depict the JAU versions with green and the sequential versions with red. Because of the significant difference in execution time, we use a logarithmic scale. Again, we used $k = 200$ iterations for all methods.

In all of our experiments the JAU versions showed important improvements in execution time, the speed-up reaching values as high as 10.6 times for NSGK, 10.8 times for SGK and 12 times for AK-SVD. This was to be expected, since JAU algorithms are naturally parallel in the atom update stage.

Chapter 6

Mixing Representation and Design Methods

In this chapter we provide an empirical investigation of the impact that the sparse representation algorithm has in DL. The spark to start this study was the somewhat disconcerting fact that, in a test problem often used in the DL community, several algorithms gave quite similar result. As explained in Chapter 4 popular DL methods use OMP in their representation stage. The immediate but apparently ignored question is if OMP is the bottleneck of DL algorithms and thus progress in the atom update stage might be masked by it. Would better representation algorithms allow to discern which atom update method is in fact superior? Would better representation algorithms cause a significant decrease of the overall error in the DL problem (2.6)? Although we cannot provide definitive answers, we try at least to gain more insight into the DL process and assess DL algorithms on a steadier ground.

We give here numerical results for the two DL problems that are used very often as benchmark: dictionary recovery and DL for sparse image representation. In the general structure of Algorithm 1, we use all the combinations of 5 representation methods (OMP, OLS, POMP, POLS, LAOLS) and 6 atom update methods (SGK, P-SGK, NSGK, P-NSGK, AK-SVD, PAK-SVD). All DL algorithms are run with the same data, in particular with the same initial dictionary. For POMP, POLS and LAOLS we took $L = s$.

6.1 Dictionary recovery

Table 6.1 show the percentages of recovered atoms for parallel and sequential AK-SVD averaging over 50 tests.

Here are some conclusions that can be drawn from the results.

Table 6.1: Percentage of recovered atoms for AK-SVD and PAK-SVD

Method	SNR			
	10	20	30	∞
AK-SVD(OMP)	10.48	93.44	93.20	93.00
AK-SVD(OLS)	12.24	93.64	94.60	95.20
AK-SVD(POMP)	54.04	**98.44**	97.64	98.12
AK-SVD(POLS)	62.56	98.12	**98.28**	**98.96**
AK-SVD(LAOLS)	48.16	96.96	96.72	96.44
PAK-SVD(OMP)	10.44	93.24	93.80	93.96
PAK-SVD(OLS)	12.12	94.16	94.12	94.72
PAK-SVD(POMP)	56.32	98.28	98.12	98.20
PAK-SVD(POLS)	**69.16**	98.36	98.12	97.52
PAK-SVD(LAOLS)	59.60	97.56	97.00	97.36

1. As expected, the more complex representation methods bring indeed an increase in performance: the recovery percentage obtained with POMP, POLS and LAOLS (especially the first two) is clearly better than with OMP, for all atom update methods. OLS and OMP are almost at the same level, with a marginal advantage for OLS.

2. For the same representation method, there is little difference between the atom update methods.

The conclusion is that the recovery test (with these commonly used data) is not relevant for comparing atom update methods and that the sparse representation is actually the bottleneck here. A reason may be the small size of the problem or the constant used to decide similarity between the recovered and original atoms. In any case, it appears that the progress in atom update methods can no longer be assessed with this test.

6.2 Dictionary learning

We built the training signals Y from $m = 8192$ random patches. With these signals, we trained dictionaries of size $n = 256$ over 200 iterations, with target sparsity $s = 8$, using again all combinations of methods. The final errors are shown in Table 6.2 while the evolution of the representation error over the number of iterations is depicted in figures 6.1–6.2.

Regarding the final error, the conclusions are mixed. With a single exception, all the other representation methods are better than OMP, often much

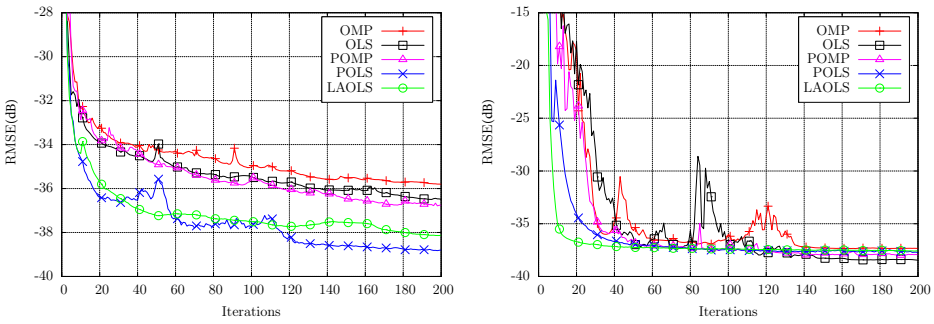

Figure 6.1: Error evolution for (P)AK-SVD with different representation methods.

Table 6.2: DL final errors

	OMP	OLS	POMP	POLS	LAOLS
SGK	0.0162	0.0150	0.0153	0.0124	**0.0117**
P-SGK	0.0136	**0.0119**	0.0126	0.0129	0.0133
NSGK	0.0154	0.0139	0.0139	0.0126	**0.0116**
P-NSGK	0.0136	**0.0116**	0.0127	0.0136	0.0141
AK-SVD	0.0162	0.0150	0.0148	**0.0114**	0.0124
PAK-SVD	0.0136	**0.0119**	0.0126	0.0130	0.0132

better; in this sense, the results are meeting normal expectations. The best results for an atom update method (in bold) are similar and OLS, LAOLS and POLS share the winners, while POMP is rather disappointing. OLS is clearly better for the parallel methods, a feature shared by OMP and POMP. On the contrary, LAOLS and POLS are systematically better for the sequential methods. We do not have an explanation for this feature of the results.

For AK-SVD (Figure 6.1) OMP provides the worst approximation but now POLS comes first with LAOLS in second place. The differences between the representation algorithms are also clearer in this graph. PAK-SVD presents error curves that are not as smooth and the differences between the sparse representations methods are not as pronounced. OLS is first and OMP is last.

POMP (Figure 6.2) bumps P-SGK and PAK-SVD first and SGK last. We notice a general smoothing effect on all the parallel update methods. POLS provides better approximations for all 6 atom algorithms and maintains the smoothing effect. It is interesting to see that AK-SVD is the best choice for POLS while, for the first time a parallel algorithm, P-NSGK, comes in last.

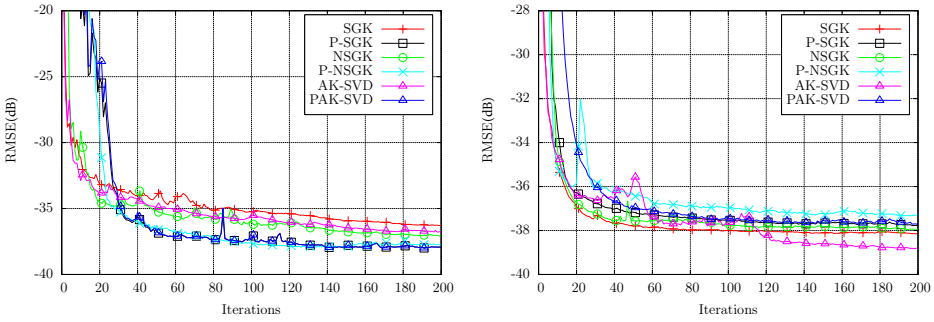

Figure 6.2: Error evolution for POMP and POLS with different dictionary update methods.

Finally, one may wonder if the designed dictionaries could be used with OMP as representation method, although another method has been employed in learning. The reason is that one may need the dictionaries in practical applications where speed is of the essence, hence one may want to use the fastest reliable representation method, i.e. OMP, which is at least a few times faster than the other methods discussed in this study. On the contrary, in learning, which is a one-time operation, we often have the luxury to use higher complexity methods for obtaining a very good dictionary. So, we computed with OMP the representations X associated the dictionaries D designed by the various methods. Remarkably, POLS gives consistently better results, which means that we could use any atom update method coupled with POLS in the DL process, then use the designed dictionary with OMP and thus get better representations than using OMP in training. We can say that this is a new design method that is better than the current methods in exactly the same conditions.

Chapter 7

Structured Dictionaries

7.1 Introduction

While the generic dictionary learning problem does not impose any specific structure on the dictionary D, some methods [15,16] build the dictionary as a union of smaller blocks consisting of ortonormal bases (ONBs) that transform the optimization problem into:

$$\begin{aligned} \underset{D,X}{\text{minimize}} \quad & \|Y - [Q_1 Q_2 \dots Q_K]X\|_F^2 \\ \text{subject to} \quad & \|x_i\|_0 \leq s, \ \forall i \\ & Q_j^T Q_j = I_p, \ 1 \leq j \leq K, \end{aligned} \tag{7.1}$$

where the union of K ONBs denoted $Q_j \in \mathbb{R}^{p \times p}$, with $j = 1 \dots K$, represents the dictionary D.

The union of orthonormal basis algorithm (UONB) [15] and the single block orthogonal (SBO) [16] algorithm enforce this structure on the dictionary by using singular value decomposition (SVD) to create each orthonormal block. The difference between the two is that for representing a single data item the former uses atoms selected via OMP from all bases, while the later uses atoms from a single orthoblock.

We are interested in parallelizing SBO because it brings data-decoupling through its single block representation system and also because it does not depend on OMP.

7.2 The P-SBO Algorithm

We term our algorithm P-SBO which is short for parallel single block orthogonal algorithm. P-SBO builds the dictionary as a union of orthoblocks.

Algorithm 4: P-SBO

`Initialization`

1 Iteratively train K_0 orthonormal blocks by randomly selecting P_0 signals from Y and applying 1ONB R times: $D = [\ Q_1 \quad \dots \quad Q_{K_0}\ ]$
2 Represent each data-item with only one of the previously computed ONBs following (7.2)

`Iterations`

3 Construct the set of the worst W represented data items and train $\tilde{K}$ new orthobases with this set. Add the new bases to the existing union of ONBs.
4 Represent each data item with one ONB
5 Train each orthobase over its new data set
6 Check stopping criterion

Each data-item from Y is constrained to use a single block Q for its sparse representation x such that $y \approx Qx$.

The representation of x results from computing the product $x = Q^T y$ and then hard-thresholding the s highest absolute value entries.

The best orthonormal base j to represent a given signal y is picked by computing the energy of the resulting representation coefficients from x and selecting the orthobase where the energy is highest. Let $E_x = \sum_{n=1}^{p} |x_n|^2$ denote the energy of a given signal x and $x^i = \text{SELECT}(Q_i^T y, s)$ denote the representation of signal y with base i, then picking the best orthonormal base can be expressed as

$$j = \underset{i=1\dots K}{\text{argmax}}(E_{x^i}). \tag{7.2}$$

Following this method, each data-item from Y is represented by a single orthobase in a process that we will call representation.

The alternative optimization iterations for performing dictionary learning on a single orthonormal base is done by the 1ONB [15] algorithm. The algorithm makes use of Procrustes orthogonalization to obtain the approximation of X and Y during training.

$$Q = UV^T \leftarrow U\Sigma V^T = \text{SVD}(YX^T) \tag{7.3}$$

Based on the above, P-SBO is described in Algorithm 4. The method is split in two parts: the initialization phase and the dictionary learning iterations.

The initialization phase builds a small start-up dictionary consisting of K_0. The resulting dictionary is used by step 2 to perform data item representation which leads to an initial sparse representation set.

The training iterations start by building $\tilde{K}$ new orthobases for the worst W represented signals using the 1ONB algorithm. Training $\tilde{K} > 1$ orthobases per iteration improves the SBO algorithm proposed in [16] by providing a better representation error and reducing the execution time. The following steps perform data-item representation and orthoblock refinement. The learning process is stopped by either reaching a given target error or the permitted maximum number of orthonormals.

7.3 Parallel SBO with OpenCL

In this section we will go through the main points behind our parallel version.

7.3.1 Parallel Representations

The sparse representations are completely independent and so their computation is done in parallel by applying (7.2) on each data-item. More specific, for each signal from Y we compute the representations with every available orthoblock and pick the one that has the highest energy.

This task fits naturally on the map-reduce model. We map the data in signal-orthobase pairs that produce the energy of the resulting sparse representation. Each pair computes the representation with the current dictionary block j ($x = Q_j^T y$), does a hard-threshold on the largest s items in absolute value, and outputs the energy E of the resulting sparse coding. We reduce the list, for each signal in Y, to the element with the largest energy leading to the choice of a single representation block.

7.3.2 Parallel Dictionary Training

Due to the decoupled nature of the data, we add parallelism at the dictionary level (each orthoblock is initialized and trained in parallel through 1ONB) and we also further parallelize the steps of each orthoblock training instance. This approach allows us to execute the sequential operations inside 1ONB (mainly the SVD routines) in parallel for each dictionary block.

n	64	96	128	160	256
$t_{learn}(s)$	366.8	396.7	416.5	438.4	642.4
$t_{rep}(s)$	0.3467	0.3753	0.8207	0.5889	2.2436
RMSE	0.0271	0.0246	0.0242	0.0230	0.0216

Table 7.1: PAK-SVD performance for $m = 32768$, $p = 64$, $s = 8$

K	8	16	24	32	64
$t_{learn}(s)$	1.8	6.7	12.3	20.9	85.4
$t_{rep}(s)$	0.0020	0.0021	0.0022	0.0021	0.0021
RMSE	0.0268	0.0245	0.0240	0.0238	0.0235

Table 7.2: P-SBO performance for $m = 32768$, $p = 64$, $s = 8$

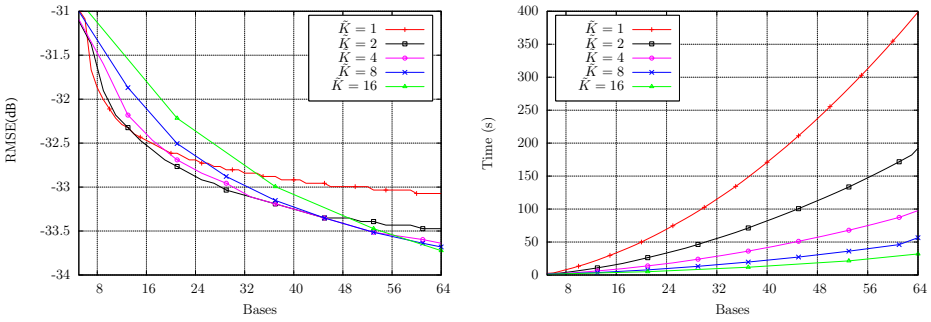

Figure 7.1: P-SBO error evolution (letf) and execution times (right).

7.4 Results and Performance

7.4.1 Execution Improvements

Tables 7.4.1 and 7.1 depict the differences in final representation error, the total time spent on dictionary learning (t_{learn}) and the time it takes to represent the data set with the final dictionary (t_{rep}).

7.4.2 Training Multiple $\tilde{K}$ Bases

We can see (Figure 7.1) that the representation error improves and drops a lot faster as we increase the number of orthobases trained at step 3 in algorithm 4. A larger $\tilde{K}$ shrinks the total number of training iterations (P-SBO steps 3–6) resulting in faster execution times.

Chapter 8

Cosparse Learning

8.1 Introduction

In the cosparse or analysis model, the overcomplete dictionary is $\Omega \in \mathbb{R}^{n \times p}$, with $n > p$, and the atoms are the rows of the dictionary. For a given signal y, the representation, now denoted $z \in \mathbb{R}^p$, is orthogonal on a set $\mathcal{I}$ of $n - s$ atoms (named cosupport), so again it lies in an s-dimensional subspace, and the representation problem is

$$\begin{aligned} \underset{z,\mathcal{I}}{\text{minimize}} \quad & \|y - z\|_2^2 \\ \text{subject to} \quad & \Omega_{\mathcal{I}} z = 0 \\ & \text{rank}(\Omega_{\mathcal{I}}) = n - s, \end{aligned} \tag{8.1}$$

where $\Omega_{\mathcal{I}}$ contains the rows of Ω with indices in $\mathcal{I}$.

Here, we propose a new dictionary training algorithm for the orthogonal case (described in Chapter 7), inspired from the cosparse DL method from [21]. The combination of techniques from both the sparse and cosparse approaches is the key to better representations and is possible due to the special characteristics induced by orthogonality.

8.2 Cosparse Orthonormal Block Training

We start with the simple remark that the sparse (4.1) and cosparse (8.1) representation problems have the same optimal error if the dictionary is orthogonal. Indeed, given D orthogonal, the problem (4.1) is solved by computing $x = D^T y$ and keeping only the largest (in absolute value) s elements, the others being forced to zero. This holds because the objective of (4.1) is equal to $\|D^T y - x\|_2^2$. For the cosparse problem (8.1), the atoms are now

Algorithm 5: 1ONB-COSP

Data: signals set Y and target sparsity s
Result: dictionary Q and sparse representations X

Initialization: Let $Q = U$ where $U\Sigma V^T = \text{SVD}(Y)$
Compute $X = Q^T Y$ and select the largest s entries of each column
foreach *atom i in dictionary Q* **do**
    Extract: $\mathcal{I} = \{j | X_{i,j} = 0\}$
    Refine: solve (8.2) to get q_i
    Update: $X = Q^T Y$ and select the largest s entries of each column
    Restructure: apply Procrustes approximation (7.3) on Y and X to orthogonalize Q

rows instead of columns, so the dictionary is $\Omega = D^T$. The two problems are connected via the relation $D^T z = x$. The $n - s$ atoms that are orthogonal on z are those corresponding to the positions of zeros in x. Otherwise said, the problem (8.1) is solved by computing $D^T y$ and setting to zero the $n - s$ smallest elements (in absolute value). We work on complementary subspaces, but the final result is the same.

8.2.1 Building Cosparse Orthonormal Blocks

UONB [15] and especially SBO [16] use 1ONB to build one orthonormal block. Using the idea behind 1ONB and concepts inspired from cosparse DL, we propose a new method for training an orthogonal block, described in Algorithm 5. Since the sparse and cosparse models are interchangeable in the orthogonal case, as explained above, we adopt an idea used for atom update in the cosparse K-SVD algorithm [21]. Denoting Q the orthogonal dictionary, an atom q_i is optimized by solving the problem

$$\begin{aligned} \underset{d_i}{\text{minimize}} \quad & \|q_i^T Y_{\mathcal{I}}\|_2^2 \\ \text{subject to} \quad & \|q_i\|_2 = 1, \end{aligned} \tag{8.2}$$

where $\mathcal{I}$ is the set of signals that do not use the atom q_i in their representation (or, taking the cosparse view, on which q_i should ideally be orthogonal). The solution of (8.2) is the singular vector corresponding to the smallest singular value of $Y_{\mathcal{I}}$. (Note the duality with sparse K-SVD, where the singular vector of the largest singular value was involved.)

We also found that further error improvement appears if, 1ONB-COSP is followed by R 1ONB training rounds. We denote 1ONB-COSP+ this

Table 8.1: Percentage of recovered atoms

s	Method	SNR			
		10	20	30	∞
3	1ONB	46.7	53.5	57.4	53.8
	1ONB-COSP	99.9	100.0	100.0	91.9
	1ONB-COSP+	100.0	100.0	100.0	99.4
4	1ONB	15.5	30.9	28.9	28.8
	1ONB-COSP	96.7	99.1	98.8	89.8
	1ONB-COSP+	98.2	99.8	99.4	97.8
5	1ONB	2.3	9.1	12.6	11.1
	1ONB-COSP	85.4	91.5	95.8	90.2
	1ONB-COSP+	91.5	95.2	98.0	95.8

succession of algorithms.

8.3 Results and Performance

We present numerical results indicating the quality improvements when using the cosparse approach.

8.3.1 Dictionary Recovery

Table 8.1 shows a big improvement in the percentages of recovered atoms, averaged over 50 runs. 1ONB-COSP is vastly superior to 1ONB. 1ONB-COSP+ improves the results where there is room for improvement, especially for larger s.

8.3.2 Dictionary Learning

In Figure 8.1 we present the average representation error over 100 runs for varying signals set sizes when using orthogonal blocks of dimension $p = 64$ ($n = 64$ atoms) with a sparsity constraint of $s = 8$. Both cosparse methods are consistent in providing a better dictionary than plain 1ONB. More so, at the cost of an increase in execution time, 1ONB-COSP+ performs better than 1ONB-COSP.

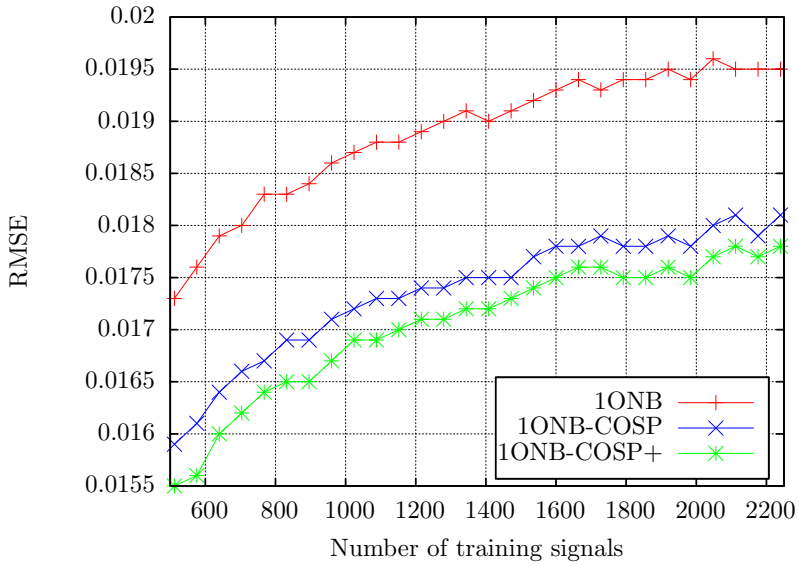

Figure 8.1: Error evolution for sparse and cosparse algorithms.

8.3.3 Unions of Orthonormal Bases with Cosparse Training

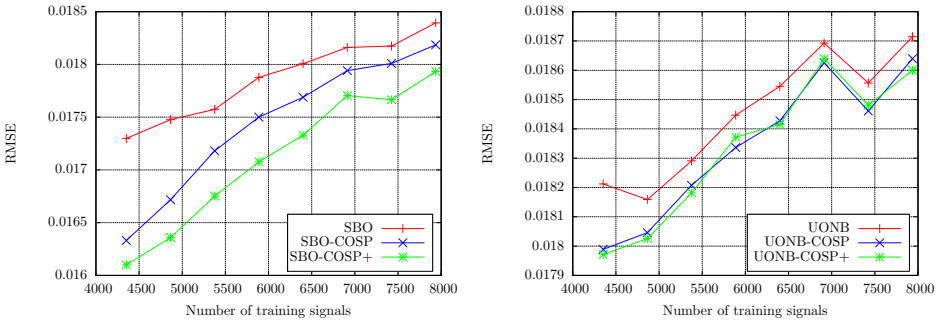

Figure 8.2: Representation error comparison of SBO and UONB variants.

To show how our cosparse approach behaves we substitute 1ONB with 1ONB-COSP or 1ONB-COSP+ in the dictionary initialization and update stage of SBO and UONB, without any other algorithmic modifications.

Chapter 9

Composite Dictionaries

9.1 Introduction

This chapter studies dictionary learning for signal denoising when using a special class of dictionaries called composite dictionaries. Given a set of signals Y that has been perturbed by a standard white gaussian noise Z, with noise level σ, and denoting with Y^c the unknown clean signals set that we want to recover, we have

$$Y = Y^c + \sigma Z. \tag{9.1}$$

If we rewrite the approximation from (5.1) as

$$Y = DX + R, \tag{9.2}$$

where R is the residual denoted as the error matrix E in (5.1), the goal of a DL denoising algorithm is to model Y^c such that the residual R from (9.2) matches the added noise:

$$Y^c \approx DX, R \approx \sigma Z. \tag{9.3}$$

There are two approaches when choosing the training set for denoising with DL: using an external signals set [22] or using an internal (or similar) set [23] of signals as the ones we want to denoise.

Here, we study the composite approach (using both internal and external sets) and apply it to the class of dictionaries structured as a union of orthonormal basis. The end result is an improvent in the quality of the denoised signals while also providing smaller execution times.

Algorithm 6: SBO-C

Data: training signals set Y^t, noisy signals Y
Result: denoised signals Y^c

Train external dictionary $E = \text{SBO}(Y^t)$
Train internal dictionary $F = \text{SBO}(Y)$
Assign each column k from Y to one orthobase $Q_E^{(k)}$ from E and one orthobase $Q_F^{(k)}$ from F following (7.2)
foreach *signal* k *from* Y **do**
    Compose: $D^c = [Q_E^{(k)} Q_F^{(k)}]$
    Represent: $X_k = \text{OMP}(D^c, Y_k)$
    Denoise: $Y_k^c = D^c X_k$

9.2 Composite Structured Dictionaries

As described in Chapter 7, SBO performs dictionary refinement independently on each block through 1ONB [15]. Even though SBO has larger memory requirements due to an increased dictionary size, its advantage over AK-SVD is the training and represntation speed while maintaining a competitive approximation quality.

9.2.1 Composing SBO with 1ONB

Our first proposal for denoising with composite structured dictionaries is to mix an SBO trained external dictionary with a specific orthoblock trained with 1ONB on the noisy set. We term this method SBO-C1.

We use SBO to train the external dictionary E on the training set Y^t and 1ONB to train the internal dictionary F on the noisy set Y.

9.2.2 Composite SBO

A natural step towards improving the representation quality of SBO-C1 is to expand the internal dictionary to more than one orthoblock. This can be achieved by performing another SBO session, this time on the noisy set, in order to create an extended internal dictionary that is still smaller than the external one. This algorithm is a direct corespondent of the composite AK-SVD method. We call it SBO-C and describe it in Algorithm 6.

Algorithm 7: Composite AK-SVD with SBO

Data: training signals set Y^t, noisy signals Y
Result: denoised signals Y^c

Train external dictionary $E = \text{AK-SVD}(Y^t)$
Train internal dictionary $F = \text{SBO}(Y)$
Assign each column k from Y to one orthobase $Q_F^{(k)}$ from F following (7.2)
foreach *signal* k *from* Y **do**
    Compose: $D^c = [E\, Q_F^{(k)}]$
    Represent: $X_k = \text{OMP}(D^c, Y_k)$
    Denoise: $Y_k^c = D^c X_k$

9.2.3 Hybrids

We also studied the case of hybrid dictionary compositions between AK-SVD and SBO.

One option is to use SBO as the external dictionary and train an AK-SVD internal block. Reversing the roles, we can train AK-SVD on the external data set and then use SBO on the internal noisy sets.

9.3 Results and Performance

Our denoising experiments were performed on images from the USC-SIPI [24] database.

We shorten AK-SVD with AK, composite AK-SVD with AK-C and the hybrid variants with S-AK where we used SBO as the external dictionary and AK-SVD as the internal one and vice-versa as AK-S.

9.3.1 Composite Dictionaries with $n = 64 + 64$

On the left panel of Figure 9.1 we show the denoising performance evolution of each method as the SNR drops. AK-S is a clear winner while plain SBO is the poorest denoiser. AK-SVD, AK-C and SBO-C1 are somewhere close in the middle presenting similar performances. Naturally, SBO-C outperforms the hybrid S-AK due to a larger internal dictionary.

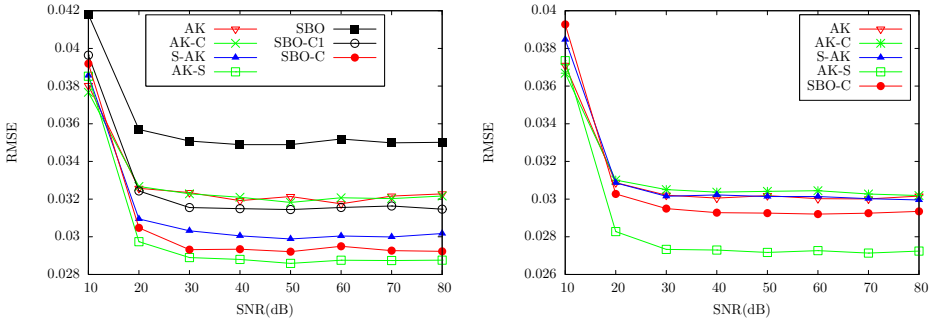

Figure 9.1: Denoising with $n = 64 + 64$ (left) and $n = 128 + 64$ (right).

9.3.2 Composite Dictionaries with $n = 128 + 64$

We present here a second experiment where we increased the size of the external dictionaries to $n = 128$ and maintained the internals at $n = 64$ leading to a fixed composite dictionary size of $n = 192$. This experiment was aimed at comparing the plain and composite AK-SVD variants with our hybrid proposals.

The right panel of Figure 9.1 shows the denoising performance as the SNR improves. We can clearly see here that AK-S is maintaining its first place position with SBO-C coming in second and AK, AK-C and AK-S fighting for third place. We can also see a plateau past the 30dB mark.

Chapter 10

Our Research

This chapter lists the articles we wrote on the subject presented in this thesis, followed by a brief description of individual contributions and the novel end results of our activity. At the end we conclude with future research.

10.1 Publications

Journal articles:

1. P. Irofti, Efficient Parallel Implementation for Single Block Orthogonal Dictionary Learning, to appear in Journal of Control Engineering and Applied Informatics, 2015, 1–8 (ISI journal with 2015 impact factor 0.537)

2. P. Irofti, Efficient Dictionary Learning Implementation on the GPU using OpenCL, to appear in U.P.B. Scientific Bulletin, series C, 2015, 1–12

Conference articles:

1. P. Irofti and B. Dumitrescu, GPU parallel implementation of the approximate K-SVD algorithm using OpenCL, in 22nd European Signal Processing Conference, 2014, 271–275 (indexed in ISI Proceedings)

2. P. Irofti and B. Dumitrescu, Overcomplete Dictionary Design: the Impact of the Sparse Representation Algorithm, in The 20th International Conference on Control Systems and Computer Science, 2015, 1–8 (indexed in ISI Proceedings)

3. P. Irofti and B. Dumitrescu, Cosparse Dictionary Learning for the Orthogonal Case, 19th International Conference on System Theory, Control and Computing, 2015, 1–6 (indexed in ISI Proceedings)

4. P. Irofti, Sparse Denoising with Learned Composite Structured Dictionaries, 19th International Conference on System Theory, Control and Computing, 2015, 1–6 (indexed in ISI Proceedings)

Under review:

1. P. Irofti and B. Dumitrescu, Overcomplete Dictionary Learning with Jacobi Atom Updates, 2015, `http://arxiv.org/abs/1509.05054`

10.2 Detailed Contributions

All of my research activity was guided by professor Bogdan Dumitrescu (B.D.). Following, I will list the contributions brought on by me in each article.

Efficient Parallel Implementation for Single Block Orthogonal Dictionary Learning At the suggestion of B.D. I implemented the SBO algorithm in OpenCL. The idea to add more than one orthonormal dictionary blocks at each iteration was mine. I performed the experiments and wrote [25].

Overcomplete Dictionary Learning with Jacobi Atom Updates After our results with K-SVD from [20], I had the idea to generalize the atom update strategy and apply it to other DL algorithms. I implemented the algorithms and performed the experiments; the article [26] was written by B.D. and I.

Efficient Dictionary Learning Implementation on the GPU using OpenCL The OpenCL DL framework was designed and implemented by me; [27] was written by me.

GPU parallel implementation of the approximate K-SVD algorithm using OpenCL The idea to parallelize K-SVD was given by B.D. We both worked on getting a feasible parallel atom update stage. The implementation and experiments were done by me and [20] was written by B.D. and I.

Overcomplete Dictionary Design: the Impact of the Sparse Representation Algorithm B.D. had idea to compare sparse representations algorithms and their impact on the dictionary update stage. I came up with

the new POLS algorithm. The implementations were done by me and B.D. and the experiments were performed by me; [28] was written by B.D. and me.

Cosparse Dictionary Learning for the Orthogonal Case The idea of applying cosparse DL for orthogonal dictionaries was mine. B.D. helped with ideas and suggestions about refining the 1ONB-COSP algorithm. The implementation and experiments were done by me. Both I and B.D. wrote the article [29].

Sparse Denoising with Learned Composite Structured Dictionaries The idea of performing denoising with composite dictionaries was mine. I implemented the algorithms, performed the experiments, and wrote [30].

10.3 Original End Results

In this section we list the finite novel products of our research.

- Theory:
  - Jacobi Atom Update framework for dictionary design that reduces representation error, improves execution times and permits full parallelism (Chapter 5)
  - progress in the atom update stage is masked when using the same sparse representation algorithm (Chapter 6)
  - using a more involved sparse representation algorithm (such as POLS) when performing DL and then switching to a faster method (like OMP) when doing representation leads to improved error minimization and execution performance (Chapter 6)
- Algorithms:
  - new sparse representation algorithm POLS that leads to better results and smoother convergence (Chapters 4 and 6)
  - new parallel dictionary update algorithms dervied from the JAU framework: PAK-SVD, P-SGK and P-NSGK (Chapters 5 and 6)
  - new parallel P-SBO algorithm for dictionaries structured as a union of orthonormal basis that improves the final error and the execution times (Chapter 7)

  - new cosparse algorithms 1ONB-COSP and 1ONB-COSP+ that significantly improve dictionary recovery and reduce representation when used as stand-alone and also within SBO and UONB (Chapter 8)
  - new denoising algorithms using composite dictionaries SBO-C1, SBO-C, S-AK and AK-S that provide sharper results with faster execution times (Chapter 9)

- Software:
  - Parallel dictionary learning library for GPUs using OpenCL that includes all the parallel algorithms for generic and structured dictionaries listed above (implementation detailed in the chapters with the corresponding algorithms)
  - Dictionary Learning library for the popular algorithms from the field implemented in C for CPUs
  - Matlab software for image denoising using composite dictionaries

10.4 Future Research

We are currently interested in finding new methods that take advantage of the orthogonal block cosparse training when learning multiple orthoblock dictionaries, researching ways in which we can adapt the representation stage to profit from the structure of composite dictionaries, and refining our parallel implementations as new OpenCL numerical libraries are made available.

The sparse representation field is relatively new with lots of hidden treasures still waiting to be found by current and future researchers. In the future we plan to take part and make the best of this quest.

Bibliography

[1] M. Elad, *Sparse and Redundant Representations: from Theory to Applications in Signal Processing*, Springer, 2010.

[2] D.L. Donoho and M. Elad, "Optimally sparse representation in general (nonorthogonal) dictionaries via l_1 minimization," *Proceedings of the National Academy of Sciences*, vol. 100, no. 5, pp. 2197–2202, 2003.

[3] D.L. Donoho and P.B. Stark, "Uncertainty principles and signal recovery," *SIAM Journal on Applied Mathematics*, vol. 49, no. 3, pp. 906–931, 1989.

[4] M. Elad and A.M. Bruckstein, "A generalized uncertainty principle and sparse representation in pairs of bases," *Information Theory, IEEE Transactions on*, vol. 48, no. 9, pp. 2558–2567, 2002.

[5] D.L. Donoho, M. Elad, and V. Temlyakov, "Stable recovery of sparse overcomplete representations in the presence of noise," *Information Theory, IEEE Transactions on*, vol. 52, no. 1, pp. 6–18, 2006.

[6] E.J. Candes and T. Tao, "Decoding by Linear Programming," *IEEE Trans. Info. Theory*, vol. 51, no. 12, pp. 4203–4215, Dec. 2005.

[7] I. Tosic and P. Frossard, "Dictionary Learning," *IEEE Signal Proc. Mag.*, vol. 28, no. 2, pp. 27–38, Mar. 2011.

[8] K. Kreutz-Delgado, J.F. Murray, B.D. Rao, K. Engan, T.W. Lee, and T.J. Sejnowski, "Dictionary learning algorithms for sparse representation," *Neural computation*, vol. 15, no. 2, pp. 349–396, 2003.

[9] R. Rubinstein, A.M. Bruckstein, and M. Elad, "Dictionaries for Sparse Representations Modeling," *Proc. IEEE*, vol. 98, no. 6, pp. 1045–1057, June 2010.

[10] K. Engan, S.O. Aase, and J.H. Husoy, "Method of optimal directions for frame design," in *IEEE Int. Conf. Acoustics Speech Signal Proc.*, 1999, vol. 5, pp. 2443–2446.

[11] M. Aharon, M. Elad, and A.M. Bruckstein, "K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation," *Signal Processing, IEEE Transactions on*, vol. 54, no. 11, pp. 4311–4322, 2006.

[12] R. Rubinstein, M. Zibulevsky, and M. Elad, "Efficient Implementation of the K-SVD Algorithm using Batch Orthogonal Matching Pursuit," *Technical Report - CS Technion*, 2008.

[13] S.K. Sahoo and A. Makur, "Dictionary training for sparse representation as generalization of k-means clustering," *Signal Processing Letters, IEEE*, vol. 20, no. 6, pp. 587–590, 2013.

[14] M. Sadeghi, M. Babaie-Zadeh, and C. Jutten, "Dictionary Learning for Sparse Representation: a Novel Approach," *IEEE Signal Proc. Letter*, vol. 20, no. 12, pp. 1195–1198, Dec. 2013.

[15] S. Lesage, R. Gribonval, F. Bimbot, and L. Benaroya, "Learning unions of orthonormal bases with thresholded singular value decomposition," in *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, March 2005, vol. 5, pp. v/293–v/296 Vol. 5.

[16] C. Rusu and B. Dumitrescu, "Block orthonormal overcomplete dictionary learning," in *21st European Signal Processing Conference*, 2013, pp. 1–5.

[17] Y.C. Pati, R. Rezaiifar, and P.S. Krishnaprasad, "Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition," in *27th Asilomar Conf. Signals Systems Computers*, Nov. 1993, vol. 1, pp. 40–44.

[18] S. Chen, S.A. Billings, and W. Luo, "Orthogonal Least Squares Methods and Their Application to Non-Linear System Identification," *Int. J. Control*, vol. 50, no. 5, pp. 1873–1896, 1989.

[19] S. Chatterjee, D. Sundman, M. Vehkapera, and M. Skoglund, "Projection-based and look-ahead strategies for atom selection," *Signal Processing, IEEE Transactions on*, vol. 60, no. 2, pp. 634–647, 2012.

[20] P. Irofti and B. Dumitrescu, "GPU parallel implementation of the approximate K-SVD algorithm using OpenCL," in *22nd European Signal Processing Conference*, 2014, pp. 271–275.

[21] R. Rubinstein, T. Peleg, and M. Elad, "Analysis K-SVD: A dictionary-learning algorithm for the analysis sparse model," *Signal Processing, IEEE Transactions on*, vol. 61, no. 3, pp. 661–677, 2013.

[22] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *Image Processing, IEEE Transactions on*, vol. 15, no. 12, pp. 3736–3745, 2006.

[23] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *Image Processing, IEEE Transactions on*, vol. 16, no. 8, pp. 2080–2095, 2007.

[24] A.G. Weber, "The USC-SIPI Image Database," 1997.

[25] P. Irofti, "Efficient parallel implementation for single block orthogonal dictionary learning," *Journal of Control Engineering and Applied Informatics*, vol. 17, no. 3, pp. 1–8, 2015.

[26] P. Irofti and B. Dumitrescu, "Overcomplete Dictionary Learning with Jacobi Atom Updates," *ArXiv e-prints*, vol. abs/1509.05054, Sept. 2015.

[27] P. Irofti, "Efficient dictionary learning implementation on the GPU using OpenCL," *U.P.B. Scientific Bulletin, Series C*, vol. 77, no. 3, pp. 1–12, 2015.

[28] P. Irofti and B. Dumitrescu, "Overcomplete dictionary design: the impact of the sparse representation algorithm," in *The 20th International Conference on Control Systems and Computer Science*, 2015, pp. 901–908.

[29] P. Irofti and B. Dumitrescu, "Cosparse dictionary learning for the orthogonal case," in *19th International Conference on System Theory, Control and Computing*, 2015, pp. 343–347.

[30] P. Irofti, "Sparse denoising with learned composite structured dictionaries," in *19th International Conference on System Theory, Control and Computing*, 2015, pp. 331–336.