Efficient Parallel Implementation for Single Block Orthogonal Dictionary Learning

Paul Irofti

Department of Automatic Control and Computers
University Politehnica of Bucharest
313 Spl. Independenţei, 060042 Bucharest, Romania

Abstract: Dictionary training for sparse representations involves dealing with large chunks of data and complex algorithms that determine time consuming tasks. In this paper we propose an improved parallel version for the single block orthogonal dictionary learning algorithm that reduces the representation error and improves the execution time. Our solution targets OpenCL capable graphical device units and focuses on full resource utilization and efficient memory partitioning.

Keywords: sparse representation, dictionary design, parallel algorithm, GPU, OpenCL

1. INTRODUCTION

The sparse representations field is the basis for a wide range of very effective signal processing techniques with numerous applications for, but not limited to, audio and image processing.

Such applications fall naturally within the realm of parallel GPU-computing due to the data size and the way the algorithms process it. When it comes to implementations, recent years have shown a tendency towards OpenCL mainly because of its portable nature and wide industry support.

In this paper, we approach the problem of training dictionaries for sparse representations by learning from a representative data set. The goal is that given a set of signals $Y \in \mathbb{R}^{p \times m}$ and a sparsity level s_0 to find a dictionary $D \in \mathbb{R}^{p \times n}$ that minimizes the Frobenius norm of the approximation error

$$E = Y - DX \tag{1}$$

where $X \in \mathbb{R}^{n \times m}$ is the associated sparse representations matrix that uses s_0 columns (or atoms) from D for sparse coding each column (or data-item) from Y.

This is a difficult problem because both the dictionary D and the sparse representations X are unknown and so existing solutions like K-SVD (Aharon et al., 2006), AK-SVD (Rubinstein et al., 2008), MOD (Engan et al., 1999), UONB (Lesage et al., 2005), SBO (Rusu and Dumitrescu, 2013), approach this as an optimization problem solved via alternative iterations. We express this as a minimization of the Frobenius norm from (1) with an l_0-norm sparsity constraint:

$$\begin{aligned} \underset{D,X}{\text{minimize}} \quad & \|Y - DX\|_F^2 \\ \text{subject to} \quad & \|x_i\|_0 \leq s_0, \ \forall i \end{aligned} \tag{2}$$

First the dictionary is fixed and the sparse representations are found by applying a greedy pursuit algorithm. While the most popular algorithm for obtaining the sparse representations seems to be Orthogonal Matching Pursuit (OMP) (Pati et al., 1993), there are other algorithms that provide a lower approximation error with an acceptable increase in complexity such as Orthogonal Least Squares (OLS)(Chen et al., 1989), Projection Based OMP (POMP) and Look-Ahead OLS (LAOLS)(Chatterjee et al., 2012).

Next, keeping the representations fixed, the dictionary is refined by updating or expanding its content. K-SVD keeps a fixed dictionary size at each iteration and updates each atom in sequence through the use of SVD after which it also updates the affected representations. Its approximate version (AK-SVD) avoids the cost of an SVD decomposition for each atom by performing one round of the power method to compute the singular vector. Given that X is fixed, MOD approaches this problem by directly solving the following l_2 minimization:

$$\|Y - DX\|_2 \tag{3}$$

A broader explanation of the problem, earlier results and applications are presented in Rubinstein et al. (2010); Tosic and Frossard (2011).

While the generic dictionary learning problem doesn't impose any specific structure on the dictionary D, some methods(Lesage et al., 2005; Rusu and Dumitrescu, 2013) build the dictionary as a union of smaller blocks consisting of ortonormal bases (ONBs) that transform the optimization problem into:

$$\begin{aligned} \underset{D,X}{\text{minimize}} \quad & \|Y - [Q_1 Q_2 \dots Q_K]X\|_F^2 \\ \text{subject to} \quad & \|x_i\|_0 \leq s_0, \ \forall i \\ & Q_j^T Q_j = I_p, \ 1 \leq j \leq K \end{aligned} \tag{4}$$

[1] This work was supported by the Romanian National Authority for Scientific Research, CNCS - UEFISCDI, project number PN-II-ID-PCE-2011-3-0400 and by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Ministry of European Funds through the Financial Agreement POSDRU/159/1.5/S/132395. E-mail: paul@irofti.net.

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) and the single block orthogonal (SBO) 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. Because of its representation strategy, SBO uses more dictionary blocks than UONB but also executes faster while maintaining the same representation error.

We are interested in parallelizing SBO because it brings data-decoupling through its single block representation system and also because it doesn't depend on OMP which raised hard full GPU occupancy problems, even when applying the partitioned global address space approach, due to its high memory footprint(Irofti and Dumitrescu, 2014).

This article presents in section 2 an improved parallel algorithm called P-SBO, followed by details of it's OpenCL implementation in section 3, and the numerical results supporting it's representation error and execution time improvements in section 4.

2. THE P-SBO ALGORITHM

P-SBO builds the dictionary as a union of orthoblocks. Each data-item from Y is constrained to use a single block Q for its sparse representation x such that:

$$y \approx Qx \tag{5}$$

The representation of x results from computing the product $x = Q^T y$ and then hard-thresholding the s_0 highest absolute value entries. This is performed through partial selection as described in algorithm 1.

Algorithm 1: SELECT

Data: unsorted list $x \in \mathbb{R}^n$, partial selection k
Result: partially sorted list x

```
1 for i ← 1 to k do
2     maxidx = i
3     maxval = x(i)
4     for j ← i + 1 to n do
5         if ‖x(j)‖ > ‖maxval‖ then
6             maxidx = j
7             maxval = x(j)
8     x(i) ↔ x(maxidx)
```

Given a list x the algorithm proceeds to do in-place sorting by finding the absolute maximum value (steps 4–7) and placing it at the top of the list (step 8). This is repeated k times (step 1) by performing the search on the remaining entries from x (steps 2–3). Even though this has an $O(kn)$ complexity, which is asymptotically inefficient, in our case k is small enough that it makes our choice sufficiently efficient and trivial to implement.

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:

$$j = \underset{i=1\dots K}{\operatorname{argmax}} \sum_{s=1}^{s_0} |Q_i^T y| \tag{6}$$

It's enough to compute the energy of the representations because the norm is preserved by the orthogonal dictionary blocks.

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

The alternative optimization iterations for performing dictionary learning on a single orthonormal base is presented in algorithm 2.

Algorithm 2: 1ONB

Data: signals set Y, initial dictionary Q_0, target sparsity s_0, number of rounds R
Result: trained dictionary Q, sparse coding X

```
1 Q = Q_0
2 for r ← 1 to R do
3     X = Q^T Y
4     X(:, j) = SELECT(X(:, j), s_0)
5     P = Y X^T
6     UΣV^T = SVD(P)
7     Q = UV^T
```

By keeping a fixed dictionary Q, step 3 computes the new representations X and step 4 performs hard-thresholding through partial sorting to select the largest s_0 values on each column. Using the new matrix X, the dictionary is refined (step 7) by using the product of the resulting orthonormal matrices from the SVD computation in step 6. This orthogonal approximation of X and Y is also called Procrustes orthogonalization in the literature.

The results from Lesage et al. (2005) show that, with a good initialization (step 1), good results can be reached by just a few iterations ($R < 5$ in step 2). Also, a good starting point when creating a new orthoblock is to use the left-hand side orthonormal matrix of the SVD decomposition of the given data set:

$$Y = U\Sigma V \rightarrow Q_0 = U \tag{7}$$

Based on the above, P-SBO is described in algorithm 3.

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 orthobases each trained with P_0 sized signal chunks that are used by 1ONB to initialize and train a new orthobase (step 1). The resulting dictionary is used by step 2 to perform data item representation which leads to an initial sparse representation set.

Algorithm 3: P-SBO

`Initialization`

1 Iteratively train K_0 orthonormal blocks by randomly selecting P_0 signals from Y and applying 1ONB R times: $D = [\, Q_1 \ \ldots \ Q_{K_0} \,]$
2 Represent each data-item with only one of the previously computed ONBs following (6)

`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

The training iterations start by building $\tilde{K}$ new orthobases for the worst W represented signals using algorithm 2 and expanding the dictionary to include the new ONBs (step 3). Training $\tilde{K} > 1$ orthobases per iteration improves the SBO algorithm proposed in Rusu and Dumitrescu (2013) by providing a better representation error and at the same time reducing the execution time as can be seen in the numerical experiments from section 4.

Given that the dictionary has changed, a new data-item representation is needed and with that step 4 computes a new set of sparse representations. Step 5 refines the dictionary D by applying 1ONB on each orthobase over its newly associated data set. The learning process is stopped by either reaching a given target error or the permitted maximum number of orthonormals.

3. PARALLEL SBO WITH OPENCL

In this section we will go through the main points behind our parallel version, then give some details on the OpenCL specifics.

3.1 Parallel representations

The sparse representations are completely independent and so their computation is done in parallel by applying (6) 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. As shown in Rusu and Dumitrescu (2013), computing the energy is enough.

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_0 items in absolute value, and outputs the energy E of the resulting sparse coding. Parallelization is done in bulk by performing the above for all ONBs at once in groups of $\tilde{m}$ signals. The result is that each data item has an associated energy list of its representation with each block from the dictionary. 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.

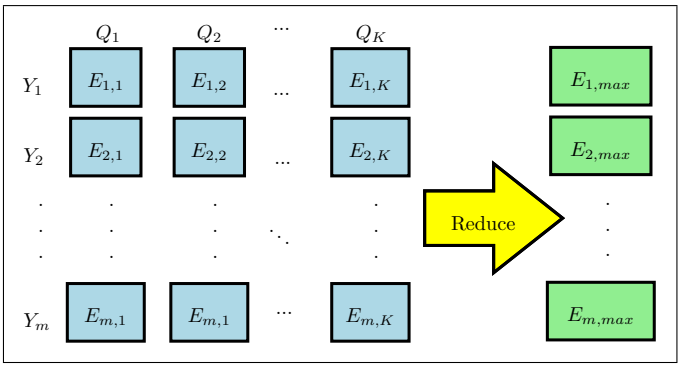

Fig. 1. MapReduce for $\tilde{m} = 1$ and K orthobases

3.2 Parallel dictionary training

Dictionary learning is performed by the operations of 1ONB described in algorithm 2. P-SBO makes use of 1ONB in three different contexts: once during the initialization phase (step 1), and twice during the training iterations while learning a new dictionary for the W worst represented signals (step 3) and while training the existing dictionary over its new data set (step 5).

Due to the decoupled nature of the data, we add parallelism at the dictionary level (each orthoblock is initialized and trained in parallel) and we also further parallelize the steps of each orthoblock training instance (see figure 2). This approach allows us to execute the sequential operations inside 1ONB (mainly the SVD routines) in parallel for each dictionary block.

If an initial orthonormal basis is not supplied, we generate a new basis by using the singular value decomposition as described in (7). This, along with the other SVD operation from step 6 are executed in parallel for each dictionary block. The alternative optimization iterations (steps 3–6) train the orthonormal dictionary Q such that $\|Y - QX\|_F$ is minimized or reduced. First, keeping a fixed dictionary, the sparse representations are computed in step 4. Since this is done via matrix multiplication of large dimensions it can be easily parallelized through the classic concurrent sub-block multiplication routines. The target sparsity is obtained by hard-thresholding the largest s_0 absolute value entries (step 4). We compute the thresholding in parallel for groups of $\tilde{m}$ signals by evenly partitioning the global address space for each thread of execution. Second, using the new matrix X, we update the dictionary via the SVD decompositon (step 6) of YX^T from step 5 by using the resulting orthonormal matrices U and V (step 7). We perform the YX^T matrix multiplication and the decomposition in parallel just as we did before. Step 7 represents a matrix multiplication of relatively small dimensions ($p \times p$) for which analysis showed that it is better to employ a partitioned global address space strategy so that each thread performs a few corresponding vector-matrix operations resulting in a simultaneous update of all orthobases.

3.3 OpenCL implementation details

The OpenCL standard abstracts hardware into processing elements (PE) that are grouped and executed in parallel on compute units (CU) located on the OpenCL device

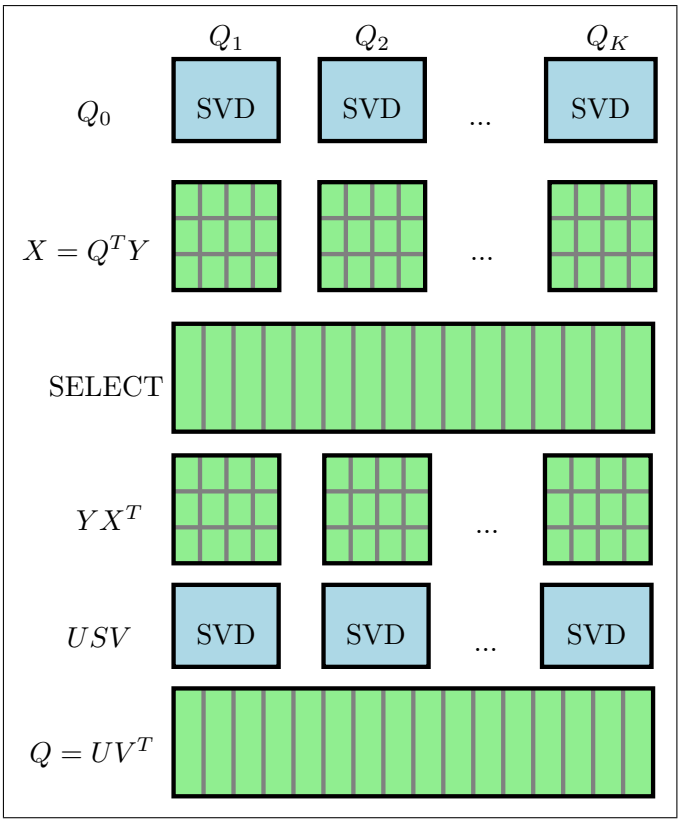

Fig. 2. The parallel execution of 1ONB for $R = 1$ rounds and K orthobases. Each block represents a task and each sub-block depicts a thread of execution within that task.

(Group, 2012). Each PE has exclusive access to private memory while sharing local memory with the PEs from the same CU and global memory with all PEs on the device.

The OpenCL execution model consists of small functions (kernels) that are executed by PEs in parallel in groups that fit within one CU. One PE is occupied by one work-item when executing one kernel. A group of PEs executing the same kernel in parallel on a CU is called a work-group.

The set of PEs available on an OpenCL device can be organized as an n-dimensional space particular to each kernel's needs. The n-dimensional space can also be split into equal subsets of PEs representing work-groups that will be scheduled for execution on the CUs available on the device.

For example, in 2D we can denote the n-dimensional range definition as NDR($\langle x_g, y_g \rangle$, $\langle x_l, y_l \rangle$). There are $x_g \times y_g$ PEs, organized on work-groups of size $x_l \times y_l$, running the same kernel. The first tuple represents the global work size (GWS) and the second the local work size (LWS).

A kernel's efficiency can be measured by looking at the resources it utilizes and the number of PEs and CUs it occupies throughout its execution. OpenCL devices execute one kernel at a time. Parallelism takes place at the CU level and it's affected by the size and the resources needed by the work-groups. For example, if local memory is exhausted before filling up all available PEs, within a CU, with work-items, then a part of the CU will idle while the rest executes the kernel leading to a sub-optimal occupancy rate.

For GPU devices, work-items from one work-group are further split by hardware and executed in parallel within sub-groups called wavefronts or waves. The number of wavefronts executed in parallel depends on the kernel usage of vector general purpose registers (VGPR), local data size (LDS) and the work-group size.

Matrix multiplication Steps 3 and 5 from the 1ONB algorithm were implemented using the BLAS library for OpenCL from AMD. The AMD kernels follow the classic GEMM BLAS model. Input matrices and the result are stored in global memory. The operation first creates matrix sub-groups and then does block-based full-matrix multiplication on them. While the AMD implementation doesn't take full advantage of the hardware underneath, it's fast enough for our use-case. We compensate it poor occupancy of the GPU resources (profiling our simulations with AMD's CodeXL showed 33.3% for the sub-grouping and 25% for the block multiplication) by scheduling as many GEMM operations at the same time as there are orthobasis (P-SBO step 1 and step 5).

Representation Given k orthoblocks, all the operations required for finding the best dictionary block for the sparse representation of each data item from the signal set, P-SBO step 2 and 4, were packed and implemented by a single OpenCL kernel following the optimization problem (6).

The input matrices as well as the resulting orthobase representation index of each signal and its energy are kept in global memory. We can keep the actual sparse representations in private memory because only the energy and base representation indices are needed by P-SBO. During representation, the sparse signal storage is accessed multiple times for each orthobase in order to compute $x = Q^T y$. Keeping the memory private gains us low latency times at the expense of an increased number of vector general purpose registers used which, in turn, leads to a lower occupancy level. Our numeric experiments showed that lower latency outbids by far a partitioned global memory, full-occupancy version of the kernel.

We designed the representation kernel following the map-reduce paradigm. We map each work-item to a signal-orthoblock couple. Each processing element is in charge of sparse coding and computing the resulting energy of a few $\tilde{m}$ signals using a single orthobase. The energy is saved in a matrix in local memory at the signal-orthobase coordinates corresponding to the work-item's position in the work-group. We keep 2-dimensional work-groups with orthobases in the first dimension and signals on the second as depicted on the left side of figure 1. And so we split the signal set in $\tilde{m}$ sized chunks representing the number of work-groups scheduled for processing on the compute-units, corresponding to an $NDR(\langle k, m \rangle, \langle k, \tilde{m} \rangle)$ splitting. The reduction on the columns of the energy matrix is performed by each work-item with ID 0 in the orthobase dimension (see the right-side of figure 1). Even though this approach leaves most of the work-items idling when reducing, the overhead of doing map-reduce in the same

kernel (opposed to doing it in two separate ones) is insignificant in this case.

This design choice and the way it affects occupancy can be observed in figure 3 that shows how resource utilization affects the number of simultaneous active wavefronts for the representation kernel. As expected, keeping the sparse representations in private memory increased the number of VGPRs used that in turn limited the number of active wavefronts to 6 as depicted in the center graph. The left and right panes show that increasing the work-group size to more than 192 work-items or expanding the LDS past 16KB would decrease the device occupancy even further.

Dictionary training The dictionary update process, P-SBO step 1 and step 5, was split into parts and implemented by multiple OpenCL kernels. We keep the input matrices for the dictionary and the signal set in global memory as well as the resulting sparse representations. The dictionary bases are modified in-place.

Before starting the dictionary training phase in P-SBO's step 5, we group the signals in blocks based on the dictionary-base used for their representations. This speeds-up the training process by using coalesced memory in P-SBO's parallel implementation. We first build a list of signals for each base Q and then we walk it contiguously copying the signals using Q overwriting the matrix Y. This is a cheap operation that brings a big performance boost by helping data access times of the execution threads. Copying proved to be up to 1000× more effective by mapping the signal matrix in host memory and using *memcpy* than plainly using *clEnqueueCopyBuffer*.

For the implementation of algorithm 2 we decided to use a Numerical Recipes based implementation of the SVD algorithm. We execute it in parallel through an OpenCL kernel for each orthoblock on the GPU with an $NDR(\langle k\rangle, \langle 1\rangle)$ splitting. The matrix multiplications (steps 3 and 5), as discussed earlier, are processed by the BLAS kernels from AMD.

The operations for partial selection (step 4 in 1ONB) were packed and implemented as a separate OpenCL kernel. The sparse signal set is kept in global memory and each work-item is in charge of doing SELECT on $\tilde{m}$ signals. Numerical experiments on our hardware pointed out that a splitting of $NDR(\langle m\rangle, \langle \tilde{m} = 256\rangle)$ gives the best performance results while keeping full GPU occupancy.

Figure 4 shows how resources limit the number of active wavefronts for the partial selection kernel. We can see that using a work-group size within 128 and 256 work-items, up to about 10 VGPRs and an LDS size of less than 10KB would permit the partial selection kernel to reach full utilization of the GPU. Our kernel is marked with a squared dot on the graphs from figure 4 and it's clearly within these limits.

Due to the small dimensions p of the block dictionaries, using the BLAS library from AMD for processing step 7 of 1ONB for each orthobase didn't cover the IO costs. For that, we implemented a custom matrix multiplication kernel that performs the operation in parallel for the entire dictionary. And so, each work-group is in charge of computing the updated orthobase corresponding to its group-id, resulting in an $NDR(\langle k \times \tilde{m}\rangle, \langle \tilde{m}\rangle)$ splitting. Work-items within a work-group are performing vectorized vector-matrix multiplication that calculate the lines of the new orthobase corresponding to their local-id. Given that $Q \in \mathbb{R}^{p\times p}$, the number of lines each work-item has to compute is given by the ratio of $p/\tilde{m}$. For p dimensioned k orthobases we found that a subunitary ratio of the form $NDR(\langle k \times \tilde{m}\rangle, \langle \tilde{m} = p \times 8\rangle)$ gives full occupancy on our GPU.

Updating the energy of the newly created sparse representations (needed in step 3 of the next P-SBO iteration for building the worst represented signals set W) is implemented by partitioning the global address space by another OpenCL kernel. The representation matrix and the associated energy set are kept in global memory. Each work-item independently computes the energy for $m/\tilde{m}$ signals with no work-group cooperation resulting in an $NDR(\langle \tilde{m}\rangle, \langle any\rangle)$ split. We found that full-occupancy is reached on our hardware by using the $NDR(\langle K \times l\rangle, \langle l = 192\rangle)$ partitioning, where K is the maximum allowed number of orthobases.

Table 1 provides an overview of the kernels n-dimensional topology and resoure utilization while performing dictionary learning with a training signals set of $m = 16384$ of size $p = 32$ each with a target sparsity $s_0 = 4$ and $K = 16$ orthoblocks.

Each column but the last represents a kernel (representation, partial selection, custom vector-matrix multiplication and energy update, respectively). The last column shows the device limits.

The table is split in two parts. The first part shows the vector GPR usage per work-item, the LDS usage per work-group, the flattened work-group size, the flattened global work size, and the number of waves per work-group, respectively for each kernel. We can see that the representation kernel uses a lot of VGPRs in comparisson with the other kernels resulting in a reduced number of waves. It's also visible that it uses the highest number of work-items due to the maping strategy described earlier. The rest of the kernels require similar resources for execution, maximizing the number of waves per work-group and thus leading to full GPU occupancy.

In the lower part of the table we can see the constraint imposed on the total number of active waves by each resource utilization: VGPRs, local memory and local work size, respectively. The last entry shows the resulting percentage of GPU occupancy. While local memory and work-group size would allow for the simultaneous execution of 16 wavefronts, the VGPRs permit only 6 out of 24 thus resulting in a 25% occupancy for the representation kernel. For the rest of the kernels the constraints permit the maximum number of waves to be executed. More so, in the case of the vector-matrix kernel the reduced number of used VGPRs would allow more active waves than the device's limit.

4. RESULTS AND PERFORMANCE

We used colored and gray scale bitmap images for the training signals, taken from the USC-SIPI (Weber, 1997)

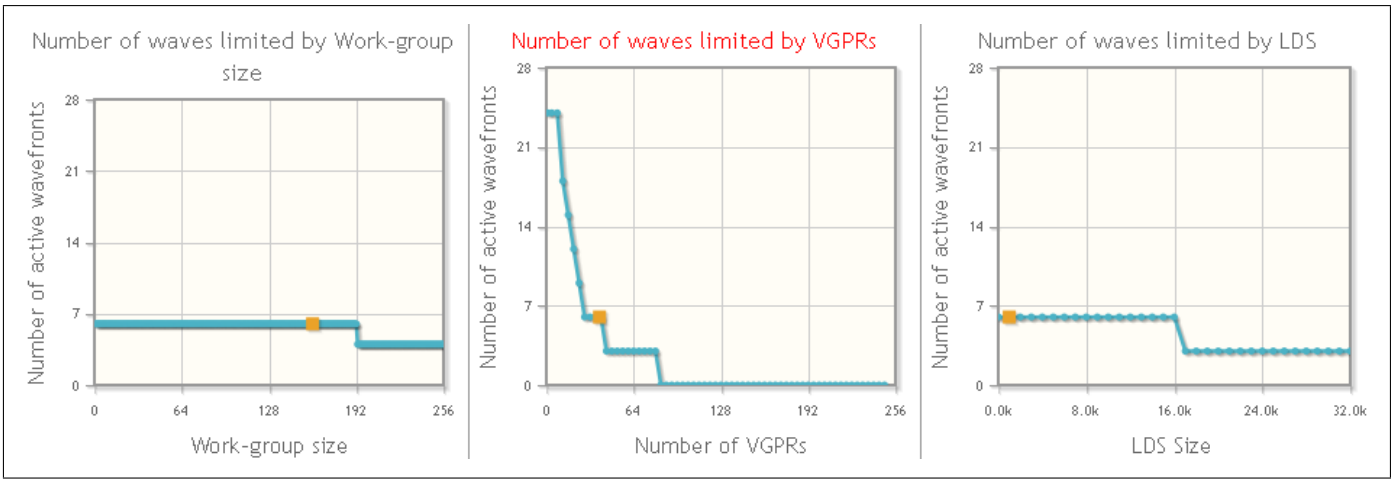

Fig. 3. Representation kernel occupancy for $K = 24$ orthobases

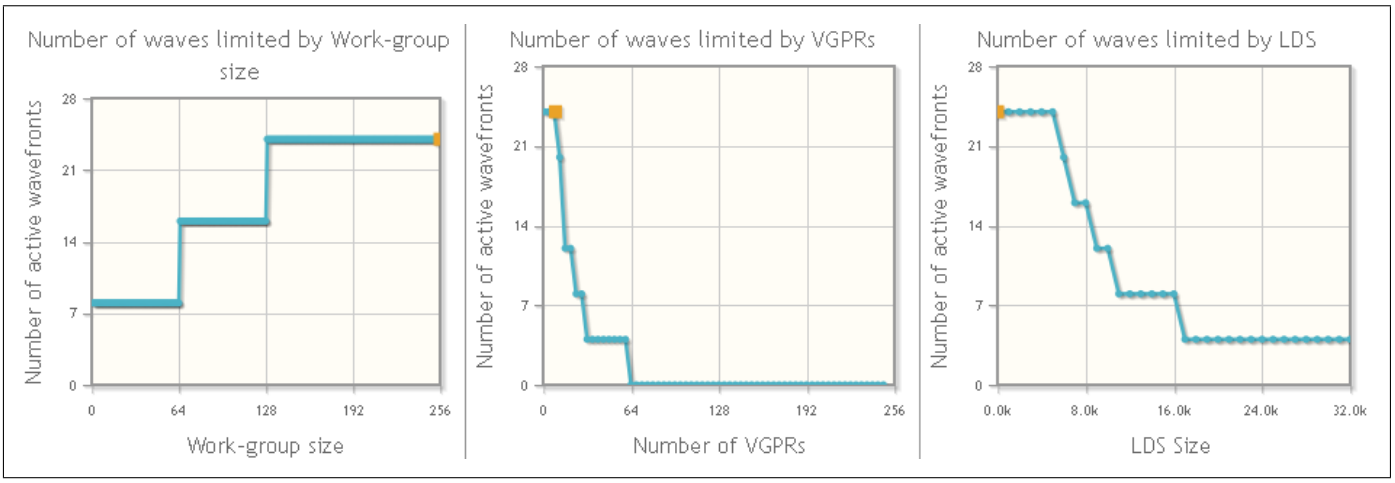

Fig. 4. Partial selection kernel occupancy for $m = 16384$ signals

Table 1. Kernel information and occupancy for $m = 16384$, $K = 16$ and $s_0 = 4$

Kernel	Rep.	Select	GEMV	Energy	Limits
VGPRs	35	9	8	4	248
LDS	1024	0	0	0	32768
LWS	80	256	256	192	256
GWS	81920	16384	1280	3072	16777216
Waves	2	4	4	3	4
VGPRs	6	24	28	24	24
LDS	16	24	24	24	24
LWS	16	24	24	24	24
Occ.(%)	25	100	100	100	100

image database (e.g. barb, lena, boat, etc.). The images were normalized and split into random 8×8 blocks

As a rule, we chose the dimensions as powers of two because this way the data objects and the work-loads are easier divided and mapped across the NDRs without the need for padding.

We tested our OpenCL implementation of P-SBO on an ATI FirePro V8800 (FireGL V) card from AMD, running at a maximum clock frequency of 825MHz, having 1600 streaming processors, 2GB global memory and 32KB local memory. Also, the CPU tests for our C implementation were made on an Intel i7-3930K CPU running at a maximum clock frequency of 3.2GHz.

4.1 Execution improvements

Tables 2 and 3 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}). We vary the total number of P-SBO orthoblocks $K = \{8, 16, 32, 64\}$ and compare with PAK-SVD instances running with a dictionary of $n = \{64, 96, 128, 256\}$ atoms and $K = 100$ iterations using full parallelization during the atoms update phase ($n = \tilde{n}$) for which the numerical simulations in Irofti and Dumitrescu (2014) gave the best representation error and the fastest execution times. We compare the resulting approximations by looking at the root mean square error

$$\text{RMSE} = \frac{\|Y - DX\|_F}{\sqrt{pm}} \tag{8}$$

which we express graphically in decibels.

While PAK-SVD can produce a slightly better error than P-SBO, the time difference is significant with P-SBO being up to 203.8 times faster than PAK-SVD at dictionary learning and 1068.4 times faster at producing sparse representations. Even though P-SBO's dictionary size is larger,

Table 2. PAK-SVD performance for $m = 32768$, $p = 64$, $s_0 = 8$ with $\tilde{n} = n$ and $K = 100$.

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 3. Parallel SBO performance for $m = 32768$, $p = 64$, $s_0 = 8$ with $K_0 = 5$ and $R = 6$

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

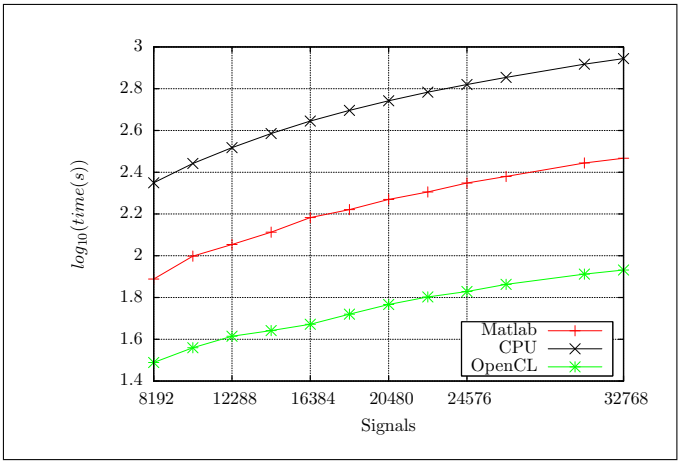

Fig. 5. Execution times for $K = 64$, $s_0 = 8$, $p = 64$.

the total memory footprint is smaller than PAK-SVD because of OMP's high memory requirements.

Turning our focus towards different P-SBO implementations, we see in figure 5 that the OpenCL implementation gives better results than the Matlab and C counterparts. Keeping a fixed number of orthonormal bases $K = 64$ and representing signal sets from as low as $m = 8192$ up to $m = 32768$, the parallel version performs 3.4 times faster than the Matlab implementation and 10.3 times faster than the single CPU C implementation.

Figure 6 describes the performance results with a fixed signal set of $m = 24576$ and a variable dictionary size starting from $K = 8$ orthoblocks up to $K = 64$. Again we can see that the OpenCL version performs a lot better than the other implementations, giving speed-ups up to 7 times.

Looking at figure 7 we see that the target sparsity s_0 doesn't really affect running times. We kept a fixed signal set $m = 32768$ and a fixed dictionary of $K = 48$, and we varied the sparsity from $s_0 = 4$ to $s_0 = 12$ on a fixed signal dimension of $p = 64$.

4.2 Training multiple $\tilde{K}$ bases

The representation error and execution improvement of P-SBO over SBO is depicted in table 4 where we varied the value of $\tilde{K}$ in the Matlab implementation of P-SBO starting from $\tilde{K} = 1$ to $\tilde{K} = 64$. We used an identical training signals set of $m = 32768$ items of size $p = 64$

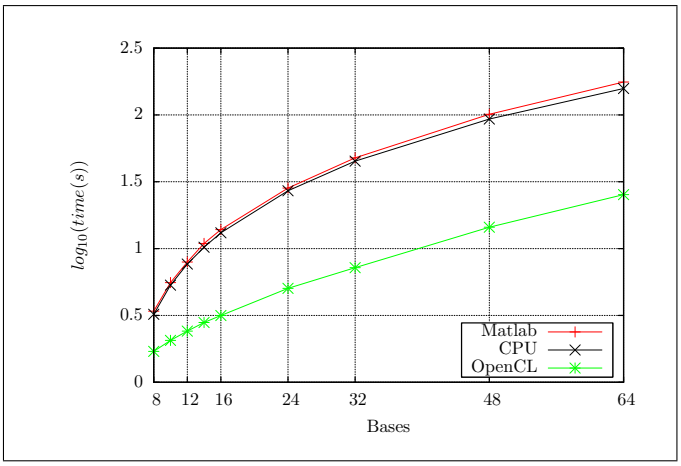

Fig. 6. Execution times for $m = 24576$, $s_0 = 4$, $p = 32$.

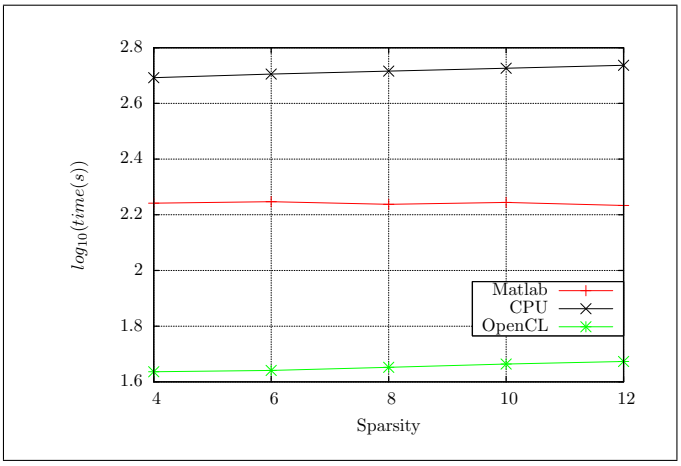

Fig. 7. Execution times for $m = 32768$, $K = 48$, $p = 64$.

Table 4. P-SBO performance for $m = 32768$, $p = 64$, $s_0 = 8$ with $K_0 = 5$, $R = 6$

$\tilde{K}$	W 8192		4096		2048	
	t(s)	RMSE	t(s)	RMSE	t(s)	RMSE
1	379	0.0222	399	0.0224	361	0.0226
2	192	0.0212	192	0.0216	187	0.0221
4	101	0.0208	98	0.0213	96	0.0217
8	57	0.0207	57	0.0213	56	0.0218
16	32	0.0206	32	0.0213	30	0.0218
32	18	0.0209	19	0.0214	18	0.0219
64	12	0.0215	12	0.0218	-	-

each with a sparsity constraint of $s_0 = 8$ and $R = 6$ ONB training rounds. The representation error is improving as $\tilde{K}$ grows until it reaches a point where the training set is too small for properly training an orthobase and so the error starts to slightly depreciate. The result is consistent with different sizes of the worst reconstruction set W.

Figure 8 shows the error evolution of the P-SBO algorithm as new bases are trained and added to the union of ONBs for different values of $\tilde{K}$. We can see that the representation error improves and drops a lot faster as we increase the number of orthobases trained at step 3 in algorithm 3.

As the number of orthobases trained for the worst-reconstructed signals set W increases the number of train-

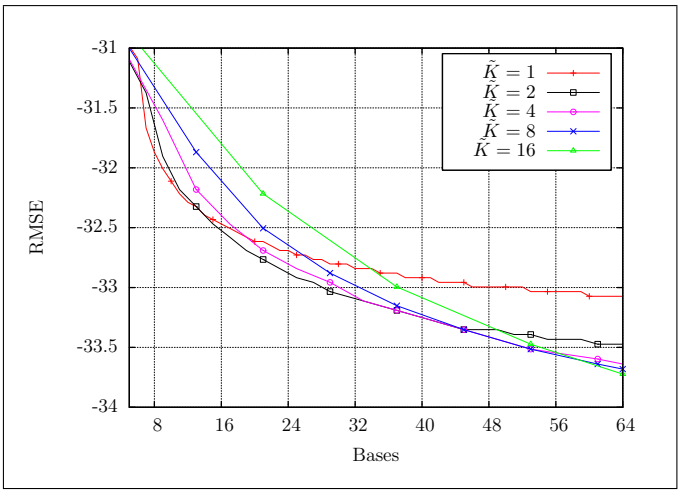

Fig. 8. P-SBO error evolution for $m = 32768$, $p = 64$, $W = 8192$, $s_0 = 8$, $K_0 = 5$, $R = 6$.

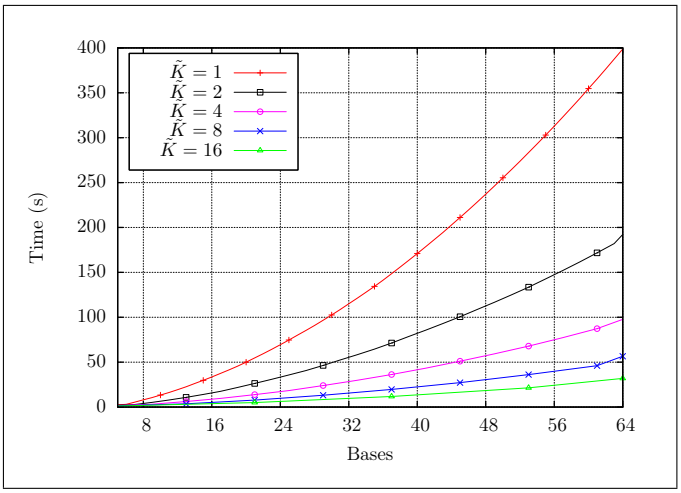

Fig. 9. P-SBO execution times for $m = 32768$, $p = 64$, $W = 4096$, $s_0 = 8$, $K_0 = 5$, $R = 6$.

ing iterations (P-SBO steps 3–6) shrinks resulting in faster execution times. This improvement is depicted in figure 9 where we show the elapsed time after each training stage when running P-SBO with the same inputs and the same constraints but with different values of $\tilde{K}$.

5. CONCLUSIONS

We provided an improved algorithm that reduces the representation error and cuts the execution time and we also proposed an efficient parallel implementation of the P-SBO algorithm. Dictionary updates are performed by refining each of the orthonormal bases concurrently. Also, we completely parallelized, in a map-reduce manner, the pursuit of finding the single best orthobase for representing a given signal. Our implementation was done in OpenCL and tested on the GPU.

Our parallel version achieves a good trade-off between algorithm complexity and data-set approximations compared to PAK-SVD due to the different representation approach and the low-memory footprint of P-SBO's representation strategy leading to better GPU occupancy confirmed in our numerical results that show a speed-up of about 200 times for dictionary learning while providing an improved representation quality. Despite its much larger dictionary size, P-SBO has a significantly lower representation time (simulations show about 1000 times speed improvement), which makes it appealing for real time applications. Also, simulations showed that P-SBO can perform about 33 times faster on the same data than SBO while also providing an improved dictionary resulting in better sparse representations.

REFERENCES

Aharon, M., Elad, M., and Bruckstein, A. (2006). K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *Signal Processing, IEEE Transactions on*, 54(11), 4311–4322. doi:10.1109/TSP.2006.881199.

Chatterjee, S., Sundman, D., Vehkapera, M., and Skoglund, M. (2012). Projection-based and look-ahead strategies for atom selection. *Signal Processing, IEEE Transactions on*, 60(2), 634–647.

Chen, S., Billings, S.A., and Luo, W. (1989). Orthogonal least squares methods and their application to non-linear system identification. *International Journal of control*, 50(5), 1873–1896.

Engan, K., Aase, S., and Husoy, J. (1999). Method of optimal directions for frame design. In *IEEE Int. Conf. Acoustics Speech Signal Proc.*, volume 5, 2443–2446.

Group, K.O.W. (2012). *The OpenCL Specification, Version 1.2, Revision 19*. Khronos Group.

Irofti, P. and Dumitrescu, B. (2014). GPU parallel implementation of the approximate K-SVD algorithm using OpenCL. In *22nd European Signal Processing Conference*, 1–5.

Lesage, S., Gribonval, R., Bimbot, F., and Benaroya, L. (2005). Learning unions of orthonormal bases with thresholded singular value decomposition. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 5, v/293–v/296 Vol. 5. doi:10.1109/ICASSP.2005.1416298.

Pati, Y.C., Rezaiifar, R., and Krishnaprasad, P.S. (1993). Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers*, 1–3.

Rubinstein, R., Bruckstein, A., and Elad, M. (2010). Dictionaries for Sparse Representations Modeling. *Proc. IEEE*, 98(6), 1045–1057.

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

Rusu, C. and Dumitrescu, B. (2013). Block orthonormal overcomplete dictionary learning. In *21st European Signal Processing Conference*, 1–5.

Tosic, I. and Frossard, P. (2011). Dictionary Learning. *IEEE Signal Proc. Mag.*, 28(2), 27–38.

Weber, A. (1997). The USC-SIPI Image Database.