Parallelizing Sparse Recovery Algorithms: A Stochastic Approach

Achal Shah* and Angshul Majumdar†
*Indian Institute of Technology, Guwahati
†Indraprastha Institute of Information Technology, Delhi

***Abstract*—This work proposes a novel technique for accelerating sparse recovery algorithms on multi-core shared memory architectures. All prior works attempt to speed-up algorithms by leveraging the speed-ups in matrix-vector products offered by the GPU. A major limitation of these studies is that in most signal processing applications, the operators are not available as explicit matrices but as implicit fast operators. In such a practical scenario, the prior techniques fail to speed up the sparse recovery algorithms. Our work is based on the principles of stochastic gradient descent. The main sequential bottleneck of sparse recovery methods is a gradient descent step. Instead of computing the full gradient, we compute multiple stochastic gradients in parallel cores; the full gradient is estimated by averaging these stochastic gradients. The other step of sparse recovery algorithms is a shrinkage operation which is inherently parallel. Our proposed method has been compared with existing sequential algorithms. We find that our method is as accurate as the sequential version but is significantly faster - the larger the size of the problem, the faster is our method.**

Keywords*—Sparse Signal Recovery, Parallel Stochastic Gradient Descent, Compressive Sensing

I. Introduction

In this work, we are interested in solving a system of linear equations where the solution is known to be sparse. Such problems arise in machine learning (regression) and in a wide class of signal processing problems (compressive sensing). There are two classes of algorithms to solve these problems - greedy approximate algorithms and optimization based techniques. Both of these approaches are inherently sequential in nature. In this work, the objective is to parallelize the optimization based techniques.

There have been a handful of attempts in the past [1], [2], [3], but all of them suffer from shortcomings (to be discussed later). However, accelerating the sparse recovery algorithms for general purpose problems is an imminent necessity for large scale problems such as medical imaging and seismic inversion problems. One cannot rely on improvements in processors to speed-up the sparse recovery algorithms. It is well known that the processors are not getting much faster; instead we have more processors that can work in parallel. One needs to harness this available 'processing parallelism' in order to accelerate the sparse recovery algorithms.

It should be kept in mind that we are not trying to run a sparse recovery algorithm in a distributed environment; i.e. with separate processor and separate memory. Rather we work in 'parallel processing shared memory' architecture. This is more realistic. For most machine learning or signal processing applications, we will have the latest PC with multiple processor cores and Graphical Processing Units (GPU) at our disposal, but will not have a cluster!

Specifically, we look at three sparse recovery problems. The first one is a simple sparse solution to a linear system of equations. The second one is a group-sparse solution to the same problem and the third one is a joint-sparse solution to the multiple measurement vector (MMV) recovery problem. The problems and their existing solutions will be discussed in detail in the next section.

Optimization based approaches to all the three problems consist of two main portions. In the first part, the gradient of the cost function is computed and the second step is a thresholding operation. Computing the gradient is a linear operation, but is not separable and hence is not naturally parallelizable. Thresholding, although a non-linear operation, is separable and hence computing it in parallel is trivial. In essence, the bottleneck is to parallelize the gradient computation. In this work, instead of attempting to directly compute the full gradient we compute stochastic gradients in multiple cores. Theory of stochastic gradient descent says that the expected stochastic gradient is same as the full gradient. Following this theory, we (empirically) compute (average) the expected stochastic gradient to estimate the actual gradient. Since, the stochastic gradients are computed using a small part of the data, and in parallel (on multiple cores of CPUs or GPUs), the gradient computation is vastly accelerated.

The rest of the paper is organized into several sections. The sparse, group-sparse and joint-sparse MMV recovery problems and their existing solutions are reviewed in the following section. The proposed methodology is discussed in section 3. We examine the experimental results in section 4. Finally the conclusions of this work are and future directions are discussed in section 5.

II. Literature Review

A. Sparse Recovery Problems

The main issue we want to address in this work is the solution to the following linear inverse problem:

$$y = Ax + \eta, \eta \sim N(0, \sigma^2) \tag{1}$$

where x is the solution we are seeking and η is the noise assumed to be Normally distributed.

Such a situation arises in machine learning as a regression problem. A direct solution via least squares is generally not a good idea since the system of equations is not well behaved always. Tikhonov regularization can be used to stabilize the solution, however such a solution lacks interpretability. In regression, the columns of A are the explanatory variables, y consists of the observations and x signifies the importance of the explanatory variables in explaining the observations. Tikhonov regularization yields a dense solution; i.e. x has non-zero values everywhere. In such a scenario, it is not possible to analyze the significant contributors (variables) to the observations. Ideally, one would like a sparse solution; i.e. x should have non-zero values only at a few locations - such a solution will explain the observations by very few variables. The Least Angle Selection and Shrinkage Operator (LASSO) [5] was proposed to achieve this goal.

$$\text{LASSO} : \min_x \|y - Ax\|_2^2 \textit{ subject to } \|x\|_1 \leq \tau \tag{2}$$

Here the constraint on the l_1-norm of the solution promotes sparsity in the solution.

Today, the problem of sparse recovery is more popular in signal processing; Compressed Sensing (CS) [6], [7] studies the problem of solving an under-determined system of linear equations when the solution is known to be sparse. In signal processing, the recovery is posed in a slightly different manner as a Basis Pursuit Denoising (BPDN):

$$\text{BPDN} : \min_x \|x\|_1 \textit{ subject to } \|y - Ax\|_2^2 \leq \varepsilon \tag{3}$$

For most signal processing problems, the noise variance (σ) is known and hence it is sensible to solve (3).

However solving the constrained problems (BPDN and LASSO) is difficult. In most practical situations, their unconstrained version is solved instead; this turns out to be a standard quadratic programming problem:

$$\textit{QP} : \min_x \|y - Ax\|_2^2 + \lambda\|x\|_1 \tag{4}$$

The three problems LASSO, BPDN and QP are equivalent for proper choices of τ, ε and λ.

The sparse recovery problem does not assume any structure to the solution. In group-sparse recovery [8], it is assumed that the indices in x are grouped, i.e. all the coefficients in a group are either non-zeroes or all are zeroes. Therefore, given the grouped indices, the problem in group-sparse recovery is to solve (1) where the solution will consist of only a few non-zero groups. The recovery is posed as follows:

$$\min_x \|y - Ax\|_2^2 + \lambda\|x\|_{2,1} \tag{5}$$

where $l_{2,1}$-norm is defined as the sum of l_2-norms of the groups. The l_2-norm over the groups in x enforces a dense solution within the selected group; but the sum over the l_2-norms act as an l_1-norm on the groups and enforces selection of only a few groups.

So far we were talking on a single measurement vector (SMV) problem (1). An extension to the SMV is the multiple measurement vector (MMV) problem:

$$Y = AX + N \tag{6}$$

where for C vectors, $Y = [y_1|\ldots|y_C]$, $X = [x_1|\ldots|x_C]$ and $N = [\eta_1|\ldots|\eta_C]$.

Joint-sparse recovery [9] is a special case, where the x_i's have a common sparse support; i.e. the non-zero values in the different x_i's occur at the same position; thus X turns out to be row-sparse. The recovery is achieved by solving the following optimization problem:

$$\min_X \|Y - AX\|_2^2 + \lambda\|X\|_{2,1} \tag{7}$$

where $\|X\|_{2,1}$ is defined as the sum of the l_2-norms over the rows of X. The idea behind the mixed norm $\|X\|_{2,1}$ is the same as before. The l_2-norm of the rows enforces non-zero values along all the elements of the selected rows, but the sum over the l_2-norms of the rows promotes sparsity in the selection of rows.

B. Recovery Algorithms

It is not possible to derive the algorithms in the limited scope of this conference paper. We will discuss the algorithms in this section; for the derivations, the interested reader should peruse the relevant references.

In this work, we focus on the Iterative Soft Thresholding algorithms for sparse recovery. We observe that all the recovery algorithms (4), (5) and (7) can be expressed in the following generic form:

$$\min_x \|y - Ax\|_2^2 + \lambda R(x) \tag{8}$$

Even (7) can be expressed in this form by rearranging A as a block diagonal matrix and concatenating the columns of X. In (8), the regularization $R(x)$ accounts for the different sparsity priors.

The first step is to express (8) in the following form, by incorporating the Landweber iterations.

$$\min_x \|b - Ax\|_2^2 + \frac{\lambda}{\alpha}R(x) \tag{9}$$

where

$$b = x_{k-1} + \frac{1}{\alpha}A^T(y - Ax_{k-1})$$

Where b is defined in each Landweber iteration based on the value of x from the previous iteration; α is the step size and usually is set at max eigenvalue (A^TA).

The equivalence of (8) and (9) is not trivial; the derivation is based on the majorization-minimization approach [10], [11]. The next step is to solve (9). This is done by taking the gradient and evaluating it to zero. The advantage of using the Landweber iteration becomes apparent here - the problem becomes separable! For example, take the concrete example of sparse recovery:

$$\min_x \|b - Ax\|_2^2 + \frac{\lambda}{\alpha}\|x\|_1 \tag{10}$$

Taking the gradient and equating it to zero, $\Delta_x\|b - Ax\|_2^2 + \frac{\lambda}{\alpha}\|x\|_1$, is the same as follows,

$$\frac{\partial}{\partial x(i)}\{(b(i) - x(i))^2 + \frac{\lambda}{\alpha}|x(i)|\} = 0$$

$$\Rightarrow x(i) - b(i) + \frac{\lambda}{2\alpha}\textit{signum}(x(i)) = 0$$

$$\Rightarrow x(i) = \textit{signum}(b(i)).\max(0, |b(i)| - \frac{\lambda}{2\alpha})$$

For the last step one should read [11]; this is called soft thresholding or shrinkage. When applied on the full vector x, this is expressed as,

$$x = \mathit{signum}(b).\max(0, |b| - \frac{\lambda}{2\alpha}) = \mathit{SoftTh}(b, \frac{\lambda}{2\alpha}) \qquad (11)$$

Thus the complete algorithm for solving the sparse recovery problem is as follows:

1) Initialize x = 0
2) At every iteration k:
   (i) Gradient update: $b = x_{k-1} + \frac{1}{\alpha}A^T(y - Ax_{k-1})$
   (ii) Soft-Threshold: $x_k = \mathit{SoftTh}(b, \frac{\lambda}{2\alpha})$

The group-sparse recovery is almost similar. The Landweber iteration remains the same. But since the regularization is different, the soft thresholding parameter is slightly modified. After Landweber iterations, the objective in each iteration is:

$$\frac{\partial}{\partial x(i)}\{(b(i) - x(i))^2 + \frac{\lambda}{\alpha}\|x_g\|_2\} = 0$$
$$\Rightarrow x(i) - b(i) + \frac{\lambda}{2\alpha}\|x_g\|_2^{-1}.|x(i)|\mathit{signum}(x(i)) = 0$$
$$\Rightarrow x(i) = \mathit{signum}(b(i)).\max(0, |b(i)| - \frac{\lambda}{2\alpha}\|x_g\|_2^{-1}.|x(i)|)$$

Here x_g denotes the group, $x(i)$ belongs to. Thus, group-sparse recovery is similar to sparse recovery; with the only difference that the threshold is modified. In sparse recovery the threshold was the same, but in group-sparse recovery it is dependent on the value of x from the previous iteration.

For joint-sparse MMV recovery the Landweber iteration is modified; the operations are on matrices.

$$B = X_{k-1} + \frac{1}{\alpha}A^T(Y - AX_{k-1})$$

Equating the derivative and setting it to zero one gets:

$$X - B + \frac{\lambda}{\alpha}\mathit{diag}(\|X^{j\rightarrow}\|_2^{-1})|X|\mathit{signum}(X) = 0$$

where j denotes the j^{th} row. The update for X is:

$$X = \mathit{signum}(B).\max(0, |B| - \frac{\lambda}{2\alpha}\mathit{diag}(\|X^{j\rightarrow}\|_2^{-1}).|X|)$$

This too is a modified soft-thresholding algorithm [12] where the threshold depends on the previous iterates.

1) Cooling: At the onset we mentioned that the BPDN and the QP are exactly the same for proper choices of parameters. Unfortunately, for most problems the relationship between λ and ε is not analytic; thus it is not possible to obtain the QP parameter λ if ε is available.

In order to address this issue, generally a cooling strategy is employed [12], [13]. We start with a high value of λ and solve the QP for the given value. In the next iteration we decrease (cool) the value of λ and solve the QP once more. This continues till the algorithm converges, i.e. till $\|y - Ax\|_2^2 \leq \varepsilon$.

C. Previous Studies

It is obvious that, all the algorithms are sequential in nature. Parallelizing them for multi-core architectures is not trivial. In [1], the iterative soft thresholding algorithm is computed on a GPU. Their parallelism is trivial. They use the GPUs to compute the matrix vector computations in the Landweber iteration; this speeds up the algorithm. This algorithm was used for Synthetic Aperture Radar (SAR) reconstruction problems. A similar approach was used in [3] as well. However, their application domain was different; sparse recovery was used for medical image reconstruction.

In [2] a parallel algorithm is derived for solving the sparse recovery problem (1); but they impose strong constraints on A. The said algorithm can only solve the sparse recovery problem if A is orthogonal! This is a very restrictive assumption and highly impractical. If the system is orthogonal, one would not employ a complicated iterative sparse recovery algorithm for recovery. One would simply apply the transpose of the orthogonal transform in order to get the desired solution! Besides, the observation model (1) is never on an orthogonal basis, either in signal processing or in regression problems of machine learning.

In [2] algorithms for speeding up greedy approximate algorithms for sparse recovery are proposed. Such algorithms are not of interest to this paper, but we discuss it nevertheless. Greedy algorithms also have two main stages - i) projection (matrix vector multiplication) and ii) support detection (of non-zero coefficients). The projection operation is simply accelerated on the GPU by taking advantage of the multiple cores; thus enhancing the matrix vector multiplication. Support detection requires sorting; this is an inherently sequential problem. However since the advent of GPUs there are parallelized versions of sorting algorithms. In [2], one such algorithm is used for support detection. The combination of GPU projection and parallelized support detection, yields high speed ups for such greedy approximate algorithms.

Sparse Recovery algorithms can also be parallelized when the matrix / operator A is isotropic, e.g. if it is restricted Fourier or wavelet kind of operator being applied on two or higher dimensional data (x). In such a situation, it is possible to parallelize the matrix vector operations of the Landweber iterations to certain extent by exploiting the isotropic properties, i.e. if one wants to compute a 2D Fourier transform, one can first apply the 1D Fourier transform along the columns first (in parallel) and then apply the 1D Fourier transform along the rows (in parallel). In a personal communication with one of the authors, Dr. Felix Herrmann of University of British Columbia's Seismic Imaging Lab, mentioned such a speed up technique that was currently being employed in the Lab's cluster.

III. Proposed Algorithm

We find that all three algorithms consist of two main parts.

1) Landweber iteration
2) Soft Thresholding / Shrinkage

The Landweber Iteration is sequential in nature; the soft thresholding is inherently parallel. Thus, parallelizing the Landweber iteration is the challenge.

To parallelize the Landweber iteration, we leverage concepts in stochastic gradient descent. We repeat the Landweber iteration for the sake of convenience:

$$b = x_{k-1} + \frac{1}{\alpha} A^T (y - Ax_{k-1})$$

This is a linear operation that can be expressed as a gradient descent step:

$$b = x_{k-1} + \frac{1}{2\alpha} \Delta_x \|y - Ax\|_2^2 \big|_{x=x_{k-1}} \tag{12}$$

The idea behind stochastic gradient descent [14], [15], [16] is to approximate the gradient of a function by a stochastic version; i.e. instead of computing the gradient on the full data, the gradient is computed on a small sample (in the extreme case only on one sample). Let g be the actual gradient (on full data) and let g_s be the stochastic gradient, then,

$$g = g_s + e \tag{13}$$

The error e is a random variable. It is proven in [14], [15], [16] that the expected error is zero $E(e) = 0$ and the convergence of the stochastic gradient to the actual gradient is determined by the second order moment $E(e^2)$.

Since Landweber iteration is a gradient descent method, the principles of stochastic gradient descent are applicable here. Therefore instead of computing the Landweber iteration on the full data, we can compute stochastic Landweber iterations on small chunks of data in parallel and then get a very close estimate of the actual Landweber update by computing the average of all the stochastic Landweber updates since the expected error asymptotically reaches 0. We replace the original Landweber update by a simple two step process:

1) Compute stochastic Landweber updates on small chunks of data by sampling y.
2) Compute the mean from the stochastic Landweber updates to estimate the full Landweber update.

The stochastic Landweber updates are computed in separate cores. The stochastic version of the Landweber iterations is given below:

1) Define N = Number of processing units available, m is the total number of samples (each row of y is considered a sample), γ is the sampling factor ($\frac{1}{N} \leq \gamma \leq 1$)
2) Randomly partition the samples, giving $\lfloor \gamma m \rfloor$ samples, along with their corresponding rows in A, to each processing unit
3) For all $i \in \{1, ..N\}$, *in parallel* do,
   $b^{(i)} = x_{k-1} + \frac{1}{\gamma}\frac{1}{\alpha} A^{(i)T}(y^{(i)} - A^{(i)} x_{k-1})$,
   where $y^{(i)}$ denotes the randomly chosen subset of the rows in y and $A^{(i)}$ denotes the corresponding rows in A
4) Aggregate from all the units, $x_k = \frac{1}{N} \sum_{i=1}^{N} b^{(i)}$
5) Return x_k

It can be seen that such a parallelization does not require explicit access to the rows or columns of the operator A. It only requires the effect of the sub-sampled rows of A. This is a stark departure from all prior attempts at using multi-core GPUs to accelerate sparse recovery algorithms.

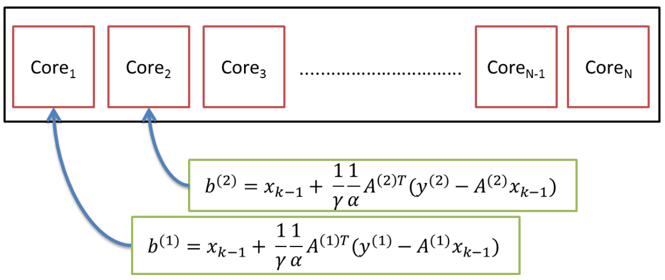
Core$_1$
Core$_2$
Core$_3$
....
Core$_{N-1}$
Core$_N$
$b^{(2)} = x_{k-1} + \frac{1}{\gamma}\frac{1}{\alpha} A^{(2)T}(y^{(2)} - A^{(2)} x_{k-1})$
$b^{(1)} = x_{k-1} + \frac{1}{\gamma}\frac{1}{\alpha} A^{(1)T}(y^{(1)} - A^{(1)} x_{k-1})$

Fig. 1. Parallelization strategy for the landweber iteration

The Landweber iteration is common to all three algorithms. The second step is the Soft-Thresholding operation, defined by,

$$SoftTh(x, T) = sign(x) \max(0, |x| - T) \tag{14}$$

Although non-linear, it simply consists of element-wise operations and is therefore, embarrassingly parallel.

The key algorithmic difference to the sparse recovery algorithm outlined in the previous section, is the replacement of gradient update and the soft-threshold operation with our proposed, parallelized alternatives.

A. Discussion

Our approach is fundamentally different from all prior studies. Previous works, leverage the speed-up in matrix vector computations on GPUs to accelerate the CS recovery algorithms. We do not rely on this phenomenon; instead of working on the full data, we divide the full data into small chunks. Each chunk is handled on separate cores. These chunks of data are independent from each other and since the chunks are small, computations on them require very less time and can occur in parallel. The computations on the small chunks are combined based on the principles of stochastic gradients.

Such an approach is hardware agnostic. Prior works could only accelerate by leveraging the fast matrix vector computations in GPUs; it couldn't exploit the multi-core CPU architectures. Our approach will work as long as there is a multi-core shared memory architecture; it does not specifically require a GPU.

The other limitation of prior studies is that they can only accelerate when the operator is available as an explicit matrix, and not when it is available as an implicit fast operator. This is rare in CS signal processing problems, where the operators are almost always implicit. In such a scenario, the prior studies fail to be useful since they have nothing to accelerate (no matrix vector products)!

IV. Experimental Evaluation

In this section, we analyze the performance of our approach for the three sparse recovery problems. The algorithms were implemented on MATLAB with multi-threading enabled, and the experiments were run on AMD Phenom II X3 710 2.6 GHz Triple-Core Processors. The proposed algorithm was parallelized on an NVIDIA GeForce GTX 680 graphics processor with 1536 CUDA cores and a standard memory of 2048 MB.

Tables 1, 2 and 3 show the comparative results between the proposed method based on stochastic gradient descent and the

TABLE I. SPARSE RECOVERY

Size	Sparsity	Proposed		Sequential		Speedup
		Time(s)	NMSE	Time(s)	NMSE	
750 × 1250	50	2.4509	1.11E-05	1.9316	1.05E-05	0.788
1500 × 2500	100	4.1659	1.20E-05	8.008	1.15E-05	1.922
3000 × 5000	200	9.484	1.30E-05	32.69	1.23E-05	3.447
4800 × 8000	320	17.223	1.37E-05	80.862	1.30E-05	4.695

TABLE II. GROUP SPARSE RECOVERY

Size	Active Groups	Proposed		Sequential		Speedup
		Time(s)	NMSE	Time(s)	NMSE	
750 × 1250	2	9.4599	3.12E-05	8.2993	2.90E-05	0.877
1500 × 2500	4	14.288	2.66E-05	23.43	2.32E-05	1.639
3000 × 5000	8	27.031	3.34E-05	84.568	2.55E-05	3.128
4200 × 7000	10	42.35	3.83E-05	166.47	2.66E-05	3.931

existing sequential algorithms for the general sparse recovery, group sparse recovery and the joint sparse MMV recovery respectively. For the group sparse recovery problem, the total number of groups is fixed at 25. For the joint sparse MMV recovery, the number of sparse coefficients is fixed at 15, and the number of measurement vectors at 10, for all experiments.

To make our experiments robust, we generated 10 i.i.d Gausssian matrices (A) for each experimental configuration. The experiment was repeated 100 times with different sparse vectors (x / X) for each of these matrices. We report both the average reconstruction accuracy (measured in terms of the Normalized Mean Squared Error) and the average reconstruction time for matrices of each size.

Tables 1, 2 and 3 conclusively prove that the proposed approach is superior to the existing, sequential algorithms. The observed NMSE values are extremely close for the proposed parallel and existing sequential versions. This validates our expectation. The speed-ups increase with an increase in the size of the matrix.

Unfortunately the speed-ups reported here are not the optimal ones. For rapid prototyping we worked on the MATLAB framework, which is restrictive. MATLAB cannot inherently parallelize the stochastic Landweber iterations on multiple cores; some parts of the implementation were sequential in spite of the best of our efforts. This has reduced the maximum possible acceleration; our proposed method can potentially accelerate even more. In a later work, we wish to implement this using CUDA C/C++ APIs (for GPUs) and OpenMP (for multicore CPUs) in order to achieve the full potential of our proposed method

V. CONCLUSION

This is the first work that accelerates general purpose sparse recovery problems on multi-core shared memory architectures. Unlike prior works [1], [3], [4], it does not use GPUs to trivially speed-up the matrix-vector products. Neither does it rely on special matrix structures (orthogonality [2], isotropic property) to make the algorithm amenable for such architectures. Moreover our method can be implemented even when the matrix A is not available as an explicit matrix, but only available as a fast operator. The prior studies require explicit matrices; they do not work for implicit fast operators on GPUs. In most signal processing (compressive sensing) problems this is infeasible. Thus, in such a situation, all previous techniques would fail and only ours would succeed.

TABLE III. JOINT SPARSE MMV RECOVERY

Size	Proposed		Sequential		Speedup
	Time(s)	NMSE	Time(s)	NMSE	
60 × 100	6.1157	2.46E-06	10.6311	2.31E-06	1.7383
60 × 200	6.7446	2.45E-06	12.0142	2.26E-06	1.7813
300 × 500	9.3509	1.04E-06	22.1004	1.03E-06	2.3634
300 × 1000	13.2061	1.09E-06	33.4678	1.06E-06	2.5343
300 × 1500	19.9587	1.33E-06	44.5388	1.28E-06	2.2315
900 × 1500	23.7138	1.03E-06	104.0274	1.02E-06	4.3868
2700 × 4500	103.6111	1.06E-06	682.4023	1.05E-06	6.5862

The major departure of our work from prior studies is a change in philosophy. Accelerating matrix-vector products is the most obvious function of GPUs, and all prior works rely on that. However we do not; actually we do not even need GPUs. Our approach can also accelerate the reconstruction algorithms on multi-core CPUs. Our philosophy is to divide the full data into small chunks and operate on them in parallel using separate cores. The results from the multiple cores are finally combined. In essence, our algorithm is more geared towards the divide and conquer regime of distributed processing.

In this work we have worked with moderately sized problems. Our proposed method can handle larger sized problems, but the sequential algorithms cannot. As we had to compare with the sequential version, we could not increase the size of the problems any further. Also, we did not parallelize too much (divided the data into 4 chunks only). Even under these circumstances (moderate size problems and small parallelism) our method beats the sequential versions in terms of speed (the accuracy remains the same). The gain in speed increases as the size of the problem increases.

This is a preliminary work and the future looks optimistic. This paves the way for further research. Using the same philosophy, we would like to parallelize analysis prior algorithms [12], [13]. A similar approach can also be used for low-rank matrix recovery problems on multi-core architectures.

REFERENCES

[1] J. Tian, J. Sun, Y. Zhang, N. Ahmad, B. Zhang, "Parallel Implementation of Compressive Sensing Based SAR Imaging with GPU", Journal of Convergence Information Technology (JCIT), Vol. 6, No. 12, pp. 122 128, 2011.

[2] A. Borghi, J. Darbon, S. Peyronnet, T. F. Chan, S. Osher, "A Simple Compressive Sensing Algorithm for Parallel Many-Core Architectures", Journal of Signal Processing Systems, Vol. 71 (1), pp 1-20, 2013.

[3] A. Shukla, A. Majumdar and R. K. Ward, "Real-Time Dynamic MRI Reconstruction: Accelerating Compressed Sensing on Graphical Processor Unit", IASTED Signal and Image Processing, 2013.

[4] J. D. Blanchard, J. Tanner, "GPU accelerated greedy algorithms for compressed sensing", Mathematical Programming Computation, Vol. 5 (3), pp 267-304, 2013.

[5] R. Tibshirani, "Regression shrinkage and selection via the lasso", J. Royal. Statist. Soc B., Vol. 58, No. 1, pp. 267-288, 1996.

[6] E. J. Cands, J. Romberg and T. Tao, Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. (IEEE Trans. on Information Theory, 52(2) pp. 489 - 509, February 2006

[7] D. Donoho, Compressed sensing. (IEEE Trans. on Information Theory, 52(4), pp. 1289 - 1306, April 2006)

[8] J. Huang and T. Zhang, "The benefit of group sparsity", Ann. Statist. Volume 38 (4), 1978-2004, 2010.

[9] E. van den Berg, M. P. Friedlander, "Joint-sparse recovery from multiple measurements", IEEE Trans. Info. Theory, Vol. 56(5), pp. 2516-2527, 2010.

[10] I. W. Selesnick and M. A. T. Figueiredo, "Signal restoration with overcomplete wavelet transforms: comparison of analysis and synthesis priors", Proceedings of SPIE, Vol. 7446 (Wavelets XIII), 2009.

[11] http://cnx.org/content/m32168/latest/

[12] A. Majumdar and R. K. Ward, "Synthesis and Analysis Prior Algorithms for Joint-Sparse Recovery", IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 3421-3424, 2012.

[13] A. Majumdar and R. K. Ward, "On the Choice of Compressed Sensing Priors: An Experimental Study", Signal Processing: Image Communication, Vol. 27 (9), pp. 10351048, 2012.

[14] D. P. Bertsekas and J. N. Tsitsiklis, "Gradient convergence in gradient methods with errors", SIAM J. Optim., Vol. 10 , pp. 627-642, 2000.

[15] M. P. Friedlander and M. Schmidt, "Hybrid deterministic-stochastic methods for data fitting", SIAM J. Sci. Comp., Vol. 34, 2012.

[16] M. P. Friedlander and G. Goh, "Tail bounds for stochastic approximation", arXiv:1304.5586, 2013.