Sparse Recovery on GPUs: Accelerating the Iterative Soft-Thresholding Algorithm

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

***Abstract*—Solving linear inverse problems where the solution is known to be sparse is of interest to both signal processing and machine learning research. The standard algorithms for solving such problems are sequential in nature - they tend to be slow for large scale problems. In the past, researchers have used Graphics Processing Units to accelerate such algorithms. But these acceleration schemes were trivial - speed-ups were achieved by computing the matrix vector products on a GPU. In this work, we propose a novel technique to accelerate a popular recovery algorithm (Iterative Soft Thresholding Algorithm - ISTA). The computational bottleneck in ISTA is in computing the gradient in every iteration. We accelerate this step by efficiently computing the gradient numerically via inexpensive updates that can be easily parallelized on the GPU. Experimental results show that the proposed method can achieve more than an order of magnitude improvement, even for moderate sized problems.**

***Keywords*—Sparse Recovery, Gradient Descent, Numerical Differentiation, Graphics Processing Units**

I. Introduction

The problem of solving a linear inverse is a classical problem in signal processing and machine learning research. For most practical scenarios, the problem is noisy and one requires solving the following problem:

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

where x is the required solution and η is the noise.

Usually such inverse problems are solved by minimizing the least squares data fidelity (owing to the Gaussian distribution of noise):

$$\min_x \|y - Ax\|_2^2 \tag{2}$$

However, when the linear system is ill-conditioned, such a solution (2) does not yield robust results. One needs to regularize the problem. The simplest regularization is a Tikhonov type regularization:

$$\min_x \|y - Ax\|_2^2 + \lambda \|x\|_2^2 \tag{3}$$

In recent times, there is a lot of interest in solving problems where the solution is known to be sparse. In such scenarios, the l_2-norm penalty does not yield the desired result; an energy minimizing penalty leads to a dense solution. To get the desired solution, one usually regularizes the least squares data fidelity term by a l_1-norm penalty:

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

In signal processing, the field of Compressed Sensing (CS) is interested in solving sparse recovery problems with the additional complication that the system is under-determined. Rather than solving the unconstrained quadratic programming (QP) problem, CS assumes that the noise variance is known and solves a constrained Basis Pursuit Denoising (BPDN) [5] problem instead:

$$\min_x \|x\|_1 \textit{ such that } \|y - Ax\|_2^2 \leq \varepsilon \tag{5}$$

Such an assumption is not realistic for the machine learning community. For sparse regression problems, the noise variance is not known; instead an estimate on the sparsity level of the solution can be assumed. This leads to the following LASSO [12] formulation:

$$\min_x \|y - Ax\|_2^2 \textit{ such that } \|x\|_1 \leq \tau \tag{6}$$

The three formulations (4), (5) and (6) are equivalent for proper choice of parameters λ, ε and τ. For most practical problems, the unconstrained formulation is preferred assuming that the regularization parameter λ is known.

One of the most popular algorithms to solve (4) is the Iterative Soft-Thresholding Algorithm (ISTA) [6]. The ISTA algorithm consists of two major steps. The first step is a Landweber iteration that is known to solve a least squares problem; the second is a thresholding step to achieve a sparse solution. There are a number of other studies that have extended the basic ISTA formulation to accelerate convergence, such as FISTA [1] and NESTA [2].

ISTA is a sequential algorithm - it is slow for large scale problems. The amount of available information has increased faster than our ability to process it. It is no longer feasible to rely on improvements in processors to speed-up matrix estimation algorithms, and as a result, we have moved towards a model where we have more processors that can work in parallel. In this paper, we address the problem of accelerating ISTA by harnessing the 'processing parallelism' of modern Graphics Processing Units (GPUs).

Parallelizing the soft-thresholding operation on a GPU is trivial. The challenge is to parallelize the Landweber iteration. There have been a handful of attempts in the past [3], [4], [10], [11] to accelerate Landweber iterations on the GPU, but they all suffer from shortcomings (discussed later). Most of them are trivial techniques that use GPUs to accelerate the matrix vector product, required during the Landweber iteration.

In this paper, we propose a novel formulation for accelerating the Landweber iteration. As mentioned earlier, this is

fundamentally a gradient descent step. We propose to compute the gradient numerically. Computing the numerical gradient can be parallelized on a GPU. But instead of blindly computing the numerical gradient on the GPU, we update the gradient in a computationally cheap fashion.

The other advantage of our proposed algorithm is the ability to solve for cost functions other than the simple l_2-norm. As an example to showcase this ability, we show how our proposed formulation can be used to solve l_1-norm data fidelity problems of the following form:

$$\min_x \|y - Ax\|_1 + \lambda\|x\|_1 \tag{7}$$

Such a problem (7) arises when the noise distribution has a sharper peak, e.g. a Laplacian distribution. Such formulations have been used in the past to remove sparse impulse noise from images [13].

The rest of the paper is organized into several sections. The existing literature is briefly reviewed in the following section. The proposed methodology is discussed in section 3. We examine the experimental results in section 4. Finally, we discuss the conclusions of this work in section 5.

II. Literature Review

Parallelizing the iterative algorithms for sparse recovery on multi-core architectures is not trivial. In [11], the Iterative Soft-Thresholding algorithm is parallelized on a GPU and is used for Synthetic Aperture Radar (SAR) reconstruction problems. Their parallelism, however, is trivial. They use the GPUs to compute the matrix vector computations in the Landweber iteration; this speeds up the algorithm. A similar approach was used in [10] but in a different application domain; sparse recovery was used for medical image reconstruction.

In [4] 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 the regression problems in machine learning.

In [4] 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. Since the advent of GPUs, however, parallelized versions of sorting algorithms have been developed. In [4], 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 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).

A recent study [9] proposed a formulation based on stochastic optimization. They presented a method to parallelize the Landweber iterations by making use of concepts in stochastic gradient descent. Instead of directly computing the full gradient, they compute stochastic gradients on small parts of the data in parallel on multiple cores. They could then closely estimate the actual gradient update (on the entire data) by computing the average of all the stochastic gradient updates, since the expected error asymptotically reaches 0. Their approach is hardware agnostic - it can be used to accelerate Landweber iteration even on multi-core CPU architectures. Our approach, however, has been tuned to exploit the massive multi-threading provided by GPU architectures, and as a result, we are able to obtain much higher speed-ups (over an order of magnitude even for moderate sized problems).

III. Proposed Algorithm

In this work, we focus on iterative methods for sparse recovery. Our initial interest is in the following unconstrained form of the inverse problem:

$$\min_x \|y - Ax\|_2^2 + \lambda\|x\|_1 \tag{8}$$

In the widely used Iterative Soft-Thresholding Algorithm (ISTA), the quadratic programming problem described above is expressed in the following form by incorporating Landweber iterations:

$$\min_x \|b - Ax\|_2^2 + \frac{\lambda}{\alpha}\|x\|_1 \tag{9}$$

where

$$b = x_{k-1} + \frac{1}{\alpha}A^T(y - Ax_{k-1}) \tag{10}$$

where b is defined in each iteration based on the value of x from the previous iteration; α is the step size and usually is set at the maximum eigenvalue of A^TA.

The equivalence of (8) and (9) is not trivial; the derivation is based on the majorization-minimization approach [8], [14]. To further solve (9), we take its gradient and equate it to zero:

$$\begin{aligned}
&\Delta_x\|b - Ax\|_2^2 + \frac{\lambda}{\alpha}\|x\|_1 = 0 \\
&\Rightarrow \frac{\partial}{\partial x^{(i)}}\{(b^{(i)} - x^{(i)})^2 + \frac{\lambda}{\alpha}|x^{(i)}|\} = 0 \\
&\Rightarrow x^{(i)} - b^{(i)} + \frac{\lambda}{2\alpha}\mathrm{signum}(x^{(i)}) = 0 \\
&\Rightarrow x^{(i)} = \mathrm{signum}(b^{(i)}).\max(0, |b^{(i)}| - \frac{\lambda}{2\alpha})
\end{aligned} \tag{11}$$

The notation $x^{(i)}$ is used to denote the i^{th} component of the vector x - we will use the same notation for the rest of the

paper. The last step discussed above is called soft-thresholding or shrinkage [14]. When applied on the full vector x, this is expressed as,

$$\begin{aligned} x &= \text{signum}(b).\max(0, |b| - \frac{\lambda}{2\alpha}) \\ &= \text{softThreshold}(b, \frac{\lambda}{2\alpha}) \end{aligned} \tag{12}$$

The complete algorithm for solving the sparse recovery problem consists of a Landweber update followed by a soft-threshold operation in each iteration. Such iterative methods that are based on thresholded Landweber iterations involve the use of both a forward as well as a backward transform - this is not always feasible as the backward operator may not be available or may be expensive.

In the remainder of this section, we develop a method to solve the sparse recovery problem that can be used even in situations where the backward operation is not feasible. We then extend our proposed method to solve the much harder problem of l_1-norm minimization.

A. Sparse Recovery by l_2-norm minimization

The Landweber iteration can be expressed as the following gradient descent step:

$$b = x_{k-1} - \frac{1}{2\alpha} \nabla_x \|y - Ax\|_2^2 \big|_{x=x_{k-1}} \tag{13}$$

Instead of computing the gradient analytically as done in (9), we compute the numerical gradient. The numerical gradient for a function $f : \mathbb{R} \to \mathbb{R}$ at a point x is given by:

$$f'(x) \approx \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h} \tag{14}$$

The gradient update for the l_2-norm minimization problem is:

$$b = x_{k-1} - \frac{1}{2\alpha} \nabla_x F(x) \tag{15}$$

where

$$F(x) = \|y - Ax\|_2^2 \tag{16}$$

Computing the gradient numerically, we have:

$$\nabla_x F(x)^{(i)} = \lim_{h_i \to 0} \frac{F(x + h_i\hat{e_i}) - F(x - h_i\hat{e_i})}{2h_i} \tag{17}$$

where $\nabla_x F(x)^{(i)}$ is the i^{th} component of the gradient vector and $\hat{e_i}$ is a vector in the standard basis - the vector with a 1 in the i^{th} coordinate and 0's elsewhere.

Simplifying the above expression, we have:

$$\begin{aligned} F(x + h_i\hat{e_i}) &= \|y - A(x + h_i\hat{e_i})\|_2^2 \\ &= (y - A(x + h_i\hat{e_i}))^T (y - A(x + h_i\hat{e_i})) \\ &= y^T y - 2y^T A(x + h_i\hat{e_i}) \\ &\quad + (A(x + h_i\hat{e_i}))^T (A(x + h_i\hat{e_i})) \\ &= y^T y - 2y^T Ax - 2h_i y^T A\hat{e_i} + (Ax)^T (Ax) \\ &\quad + 2h_i (A\hat{e_i})^T (Ax) + h_i^2 (A\hat{e_i})^T (A\hat{e_i}) \end{aligned} \tag{18}$$

and similarly,

$$\begin{aligned} F(x - h_i\hat{e_i}) &= \|y - A(x - h_i\hat{e_i})\|_2^2 \\ &= (y - A(x - h_i\hat{e_i}))^T (y - A(x - h_i\hat{e_i})) \\ &= y^T y - 2y^T A(x - h_i\hat{e_i}) \\ &\quad + (A(x - h_i\hat{e_i}))^T (A(x - h_i\hat{e_i})) \\ &= y^T y - 2y^T Ax + 2h_i y^T A\hat{e_i} + (Ax)^T (Ax) \\ &\quad - 2h_i (A\hat{e_i})^T (Ax) + h_i^2 (A\hat{e_i})^T (A\hat{e_i}) \end{aligned} \tag{19}$$

Before we use the above expressions for the computation of an element of the gradient vector, we observe that not every term needs to be re-computed in each iteration - certain terms can be pre-computed and kept in memory. The obtained expressions can be reduced to:

$$\begin{aligned} F(x + h_i\hat{e_i}) = t_1 - 2y^T Ax - t_{2_i} + (Ax)^T (Ax) \\ + 2h_i (A\hat{e_i})^T (Ax) + t_{3_i} \end{aligned} \tag{20}$$

and similarly,

$$\begin{aligned} F(x - h_i\hat{e_i}) = t_1 - 2y^T Ax + t_{2_i} + (Ax)^T (Ax) \\ - 2h_i (A\hat{e_i})^T (Ax) + t_{3_i} \end{aligned} \tag{21}$$

where terms t_1, t_{2_i} and t_{3_i} can be pre-computed for each position i in the gradient vector.

The advantage of computing the numerical gradient now becomes apparent - a backward transform is no longer needed. At first this method seems suboptimal as it appears that we would need to compute the elements of the gradient vector one at a time. While each element of the gradient vector can be efficiently computed using the above mentioned pre-computations and vectorization techniques, computing the entire gradient vector in each iteration may be infeasible for higher-dimensional vectors. This is where the numerical gradient offers a second advantage - each element of the gradient vector can be computed independently and, potentially, in parallel.

The proposed method consists of four major steps in each iteration - the forward operation, matrix-vector products, gradient update and soft-thresholding. Matrix-vector products and soft-thresholding are inherently parallelizable on a GPU. The bottleneck here is the gradient computation. By computing the gradient numerically, we can avoid using the backward operator and each element of the gradient vector can be computed efficiently and in parallel using the multi-threaded CUDA architecture. Our method can therefore be applied to any operator whose forward operation can be implemented on a GPU.

B. Sparse Recovery by l_1-norm minimization

We now address the following optimization problem:

$$\min_x \|y - Ax\|_1 + \lambda \|x\|_1 \tag{22}$$

The gradient update for the l_1-norm minimization problem is:

$$b = x_{k-1} + \frac{1}{2\alpha} \nabla_x F(x) \tag{23}$$

where

$$F(x) = \|y - Ax\|_1 \tag{24}$$

Extending our proposed approach, we have:

$$\begin{aligned} F(x + h_i\hat{e_i}) &= \|y - A(x + h_i\hat{e_i})\|_1 \\ &= \|y - Ax - h_iA\hat{e_i}\|_1 \\ &= \sum_j |y^{(j)} - (Ax)^{(j)} - h_i(A\hat{e_i})^{(j)}| \end{aligned} \quad (25)$$

and similarly,

$$\begin{aligned} F(x - h_i\hat{e_i}) &= \|y - A(x - h_i\hat{e_i})\|_1 \\ &= \|y - Ax + h_iA\hat{e_i}\|_1 \\ &= \sum_j |y^{(j)} - (Ax)^{(j)} + h_i(A\hat{e_i})^{(j)}| \end{aligned} \quad (26)$$

Computing the gradient numerically (omitting the $\lim$ sign for the sake of convenience):

$$\begin{aligned} \nabla_x F(x)^{(i)} &= \frac{F(x + h_i\hat{e_i}) - F(x - h_i\hat{e_i})}{2h_i} \\ &= \frac{\sum_j |y^{(j)} - (Ax)^{(j)} - h_i(A\hat{e_i})^{(j)}|}{2h_i} \\ &\quad - \frac{\sum_j |y^{(j)} - (Ax)^{(j)} + h_i(A\hat{e_i})^{(j)}|}{2h_i} \end{aligned} \quad (27)$$

Computing the gradient vector is not trivial - each element of the gradient vector requires computing the l_1-norm of a vector. We show that it can be parallelized efficiently on a GPU by using a vectorization technique.

Let us construct a matrix P, such that:

$$P^{(i,j)} = \frac{|y^{(j)} - (Ax)^{(j)} - h_i(A\hat{e_i})^{(j)}|}{2h_i} \quad (28)$$

Computing each element of P involves only scalar operations - it can be done in constant time. Moreover, all the elements of P can be computed in parallel as they are independent of each other. The matrix P can hence be very efficiently computed using the massively multi-threaded CUDA model.

We could similarly construct a matrix Q, such that:

$$Q^{(i,j)} = \frac{|y^{(j)} - (Ax)^{(j)} + h_i(A\hat{e_i})^{(j)}|}{2h_i} \quad (29)$$

The gradient vector can then be computed using the following inherently parallel operation:

$$\nabla_x F(x) = (P - Q)\vec{\mathbf{1}} \quad (30)$$

where $\vec{\mathbf{1}}$ is a vector of all 1's.

IV. Experimental Evaluation

In this section, we analyze the performance of our approach for the sparse recovery problem. 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 algorithms were parallelized on an NVIDIA GeForce GTX 680 graphics processor with 1536 CUDA cores and a standard memory of 2048 MB.

The experimental results have been discussed in Tables 1 and 2. To ensure the robustness of our experiments, we generated 5 i.i.d Gaussian matrices (A), for each experimental configuration. The experiment was repeated 100 times with different sparse vectors 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.

TABLE I. Sparse Recovery: l_2-norm minimization

Size	Sparsity	Standard ISTA (CPU)		Proposed L-2 (GPU)		Speedup
		Time(s)	NMSE	Time(s)	NMSE	
750 × 1250	50	3.1	4.93E-06	1.8	4.96E-06	1.72
1500 × 2500	100	14.9	5.77E-06	5.6	5.76E-06	2.66
3000 × 5000	200	58.4	6.27E-06	8.4	6.10E-06	6.95
4800 × 8000	320	149.5	6.59E-06	11.5	6.63E-06	13
6000 × 10000	400	232.03	7.27E-06	16.16	7.03E-06	14.36

TABLE II. Sparse Recovery: l_1-norm minimization

Size	Sparsity	Proposed L-1 (GPU)		Speedup (vs ISTA)
		Time(s)	NMSE	
750 × 1250	50	4.7	3.70E-02	0.66
1500 × 2500	100	7.7	2.70E-02	1.94
3000 × 5000	200	17	1.80E-02	3.44
4800 × 8000	320	28.5	1.40E-02	5.25
6000 × 10000	400	41.5	1.33E-02	5.59

Table 1 presents a comparative evaluation of our proposed algorithm accelerated ISTA (aISTA) for l_2-norm data fidelity (4) with the sequential ISTA. Even on problems of moderate size, we find the acceleration to be commendable, without a deterioration in the reconstruction accuracy. We achieve an acceleration of over an order of magnitude (around 14 times) for the two larger matrices used in our experiments, and we suspect that the speed-ups would continue to increase as the size of the data approaches what is typically used in practice.

In Table 2, we present the empirical accuracy of our proposed algorithm for the l_1-norm data fidelity (7), and compare the time taken for reconstruction with that of the sequential ISTA. For the two larger problems, we observe that our method is able to accelerate sparse recovery by more than 5 times.

These results suggest that the proposed algorithm is superior to the existing, sequential approach. The proposed method is faster, even for moderate sized systems. We expect that for larger problems, the acceleration will be more pronounced.

V. Conclusion

In this work we have proposed a new approach to solve sparse recovery problems. Unlike prior works, it does not use a GPU to trivially speed-up the matrix-vector products required in the Landweber iteration. Our method hinges on computing the numerical gradient of the cost function in a parallel fashion. The goal of our work was to show how this approach can accelerate the sequential sparse recovery algorithms. We achieve more than an order of magnitude acceleration for moderate size problems over the sequential approach.

In our experiments we only considered the Iterative Soft-Thresholding Algorithm, which is the most commonly used algorithm for sparse recovery problems. There exist various other algorithms such as FISTA and NESTA that are based on the ISTA approach. Our proposed method can be used to accelerate these algorithms as well. In short, any algorithm that uses a Landweber iteration can benefit from our proposed acceleration technique. Algorithms for multiple measurement vector (MMV) problems such as the row-sparse multiple measurement vector recovery algorithms [7] can also be accelerated using our method. In the future, we plan to extend our approach to structured sparsity problems such as group-sparse recovery and row-sparse MMV recovery.

The added benefit of our approach is its ability to handle cost functions other than the l_2-norm. This is because the gradient is computed numerically. As an example, we have showed how the l_1-norm data fidelity problem can be handled with our proposed approach. But even for the l_1-norm data fidelity, we solved a linear system of equations. In the future, we would like to extend the formulation for non-linear sparse recovery problems, such as exponential or logarithmic functions instead of linear functions.

REFERENCES

[1] A. Beck and M. Teboulle. "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems", SIAM Journal on Imaging Sciences, Vol. 2, No. 1, pp. 183-202, 2009

[2] S. Becker, J. Bobin, and E. J. Candes. "NESTA: a fast and accurate first-order method for sparse recovery", SIAM Journal on Imaging Sciences 4(1), 1-39

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

[4] 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.

[5] S. S. Chen, D. L. Donoho, and M. A. Saunders. "Atomic decomposition by basis pursuit", SIAM Review, 43, 2001

[6] I. Daubechies and M. Defrise, and C. De Mol. "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint", Communications on Pure and Applied Mathematics Vol. 4 (57), 1413-1457, 2004.

[7] 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

[8] 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.

[9] A. Shah and A. Majumdar. "Parallelizing Sparse Recovery Algorithms: A Stochastic Approach", International Conference on Digital Signal Processing, 2014.

[10] 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.

[11] 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.

[12] R. Tibshirani. "Regression shrinkage and selection via the lasso", Journal of the Royal Statistical Society, Series B, Vol. 58, No. 1, pp. 267-288, 1996.

[13] B. Wohlberg and P. Rodrguez. "An L1-TV algorithm for deconvolution with salt and pepper noise", IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 1257-1260, 2009

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