PARALLEL GPU IMPLEMENTATION OF NULL SPACE BASED ALTERNATING OPTIMIZATION ALGORITHM FOR LARGE-SCALE MATRIX RANK MINIMIZATION

Katsumi Konishi

Department of Computer Science, Kogakuin University, Tokyo Japan
email: konishi@kk-lab.jp

ABSTRACT

This paper provides an alternating optimization algorithm for large-scale matrix rank minimization problems and its parallel implementation on GPU. The matrix rank minimization problem has a lot of important applications in signal processing, and several useful algorithms have been proposed. However most algorithms cannot be applied to a large-scale problem because of high computational cost. This paper proposes a null space based algorithm, which provides a low-rank solution without computing inverse matrix nor singular value decomposition. The algorithm can be parallelized easily without any approximation and can be applied to a large-scale problem. Numerical examples show that the algorithm provides a low-rank solution efficiently and can be speed up by parallel GPU computing.

Index Terms— Compressed sensing, matrix rank minimization, matrix recovery, parallel computing, GPU computing.

1. INTRODUCTION

This paper deals with the affine matrix rank minimization problem, which is the problem of finding the lowest rank matrix satisfying affine constraints. While this problem has combinatorial complexities and is NP hard in general, it has a lot of applications such as model order reduction [1], matrix completion, collaborative filtering [2], image inpainting [3] and video inpainting [4]. Therefore it is important to provide a practical algorithm to obtain a low-rank solution.

There are several optimization approaches such as the nuclear norm minimization approach [1, 5], PowerFactorization approach [6], a singular value thresholding (SVT) algorithm [7], and the fixed point continuation algorithm (FPC) [8]. While these algorithms have good performance to provide a low-rank solution, they cannot be applied to large size problems because of high computational cost derived from the calculation of inverse matrix or singular value decomposition (SVD).

This paper proposes an alternating optimization algorithm for the rank minimization problem, where a low-rank solution is obtained by maximizing the nullity of the matrix to recover. The advantage of the algorithm is that it can be speed up significantly by GPU computing and requires only computation of addition and multiplication but no inverse matrix nor SVD, which implies that the algorithm can be parallelized easily and efficiently. Parallel algorithms are very important for a large-scale problem in GPU computing because GPU has limited memory capacity. This paper provides a parallel algorithm for a large-scale matrix rank minimization problem and its parallel implementation on GPU.

2. MAIN RESULTS

2.1. Affine Rank Minimization Problem

This paper deals with the following matrix rank minimization problem,

$$\text{Minimize } \mathbf{rank}X \quad \text{subject to } \mathcal{A}(X) = \boldsymbol{b}, \tag{1}$$

where $X \in \boldsymbol{R}^{m\times n}$ is a variable matrix to be recovered, $m \leq n$, $\mathcal{A} : \boldsymbol{R}^{m\times n} \to \boldsymbol{R}^p$ is a given linear operator, and $\boldsymbol{b} \in \boldsymbol{R}^p$ is a constant vector. A special case of this problem is the matrix completion problem as follows,

$$\text{Minimize } \mathbf{rank}X \quad \text{subject to } X_{ij} = M_{ij},\ \forall (i,j) \in \mathcal{I}, \tag{2}$$

where $M \in \boldsymbol{R}^{m\times n}$ is a given matrix, and $\mathcal{I}$ is a given set of matrix indices. The problem of minimizing the matrix rank is equal to the problem of maximizing its nullity, which can be formulated as the following matrix rank maximization problem,

$$\text{Maximize } \mathbf{rank}W \quad \text{subject to } XW = \mathbf{0}_{m,n},\ \mathcal{A}(X) = \boldsymbol{b}, \tag{3}$$

where $W \in \boldsymbol{R}^{n\times n}$ and $X \in \boldsymbol{R}^{m\times n}$ are variable matrices, and $\mathbf{0}_{m,n}$ denotes the $m \times n$ zero matrix. The matrix rank minimization problem (1) is equal to the matrix rank maximization problem (3), that is, X^* is the minimizer of (1) if and only if X^* is the optimal solution of (3).

2.2. Null Space Based Alternating Optimization Algorithm

The problem (3) has difficulties about noncovexity of the objective function and the constraint $XW = \mathbf{0}_{m,m}$. This paper

Algorithm 1 NSAO algorithm.

Input: $X \in \boldsymbol{R}^{m\times n}$, γ, η
  repeat
    $W \leftarrow \arg\min_{W_{ii}=1} f_\gamma(W, X)$.
    $X \leftarrow \arg\min_{\mathcal{A}(X)=\boldsymbol{b}} f_\gamma(W, X)$.
    $\gamma \leftarrow \gamma/\eta$.
  until termination criterion is satisfied
Output: low-rank solution X

relaxes these difficulties and provides an approximate algorithm.

First, we relax the objective function to a convex function. Without loss of generality we can append the constraints $W_{ii} = 1$ for $i = 1, 2, \ldots n$ to (3). This paper proposes the following relaxed problem,

$$\begin{array}{ll} \text{Minimize} & \|W\|_F^2 \\ \text{subject to} & W_{ii} = 1,\ XW = \mathbf{0}_{m,n},\ \mathcal{A}(X) = \boldsymbol{b}, \end{array} \tag{4}$$

where W and X are variable matrices, and $\|\cdot\|_F$ denotes the Frobenius norm of the matrix. The key idea of this paper is relaxing the rank maximization problem by the Frobenius norm minimization. If there is no constraint except for $W_{ii} = 1$, we can exactly maximize the rank of W by minimizing its Frobenius norm. Therefore it is expected that (4) provides a good solution of (3) and rarely gives a lower rank solution.

Although (4) has a convex objective function, it still involves combinatorial difficulty caused by the constraint $XW = \mathbf{0}_{m,n}$. To relax this difficulty, this paper applies the Lagrangian relaxation, and the following problem is obtained.

$$\begin{array}{ll} \text{Minimize} & f_\gamma(W, X) \\ \text{subject to} & W_{ii} = 1,\ \mathcal{A}(X) = \boldsymbol{b}, \end{array} \tag{5}$$

where the function $f_\gamma(W, X)$ is defined as

$$f_\gamma(W, X) = \gamma\|W\|_F^2 + \|XW\|_F^2$$

for $\gamma > 0$. Based on (5), this paper proposes the null space based alternating optimization (NSAO) algorithm for the rank minimization problem as shown in Algorithm 1, where $\eta > 1$, and W and X are optimized alternatingly. In this algorithm, we adopt the simple update scheme for Lagrange multiplier γ, which experimentally has a good performance with few computational cost.

2.3. Gradient Projection Method

In NSAO, W and X are obtained by solving the convex quadratic programmings subject to linear equalities, where higher computational cost is required than solving the unconstrained convex quadratic programming, and the optimal solutions cannot be obtained in practical time. To reduce the computational cost, an iterating algorithm is proposed based on the traditional gradient projection method (GPM).

Algorithm 2 NSAO-GPM

Input: $X \in \boldsymbol{R}^{m\times n}$, γ, η
  repeat
    $D_\Phi \leftarrow \gamma W + X^T XW;\ F_\Phi \leftarrow P_\Phi\left(W - 2D_\Phi\right) - W$
    $\alpha_\Phi \leftarrow -\mathbf{Tr}\left(D_\Phi F_\Phi\right)/\|X^T F_\Phi\|_F^2$
    $W \leftarrow W + \alpha_\Phi F_\Phi$
    $D_\Omega \leftarrow XWW^T;\ F_\Omega \leftarrow P_\Omega\left(X - 2D_\Omega\right) - X$
    $\alpha_\Omega \leftarrow -\mathbf{Tr}\left(D_\Omega^T F_\Omega\right)/\|F_\Omega W\|_F^2$
    $X \leftarrow X + \alpha_\Omega F_\Omega$
    $\gamma \leftarrow \gamma/\eta$
  until termination criterion is satisfied
Output: low-rank solution X

Define Ω and Φ as

$$\Omega = \{X \in \boldsymbol{R}^{m\times n} :\ \mathcal{A}(X) = \boldsymbol{b}\},$$

and

$$\Phi = \{W \in \boldsymbol{R}^{n\times n} :\ W_{ii} = 1,\ \forall i\},$$

respectively, and let P_Ω and P_Φ denote the orthogonal projection on Ω and Φ, respectively. We have

$$\begin{aligned} \frac{\partial}{\partial X} f_\gamma(W, X) &= 2XWW^T, \\ \frac{\partial}{\partial W} f_\gamma(W, X) &= 2\left(\gamma W + X^T XW\right), \end{aligned}$$

and therefore Algorithm 2 is obtained based on GPM. In Algorithm 2, step length α_Φ and α_Ω are determined as follows. Because $f_\gamma(W, X + \alpha_\Omega F_\Omega)$ is a convex quadratic function of α_Ω for given W and X and because it holds that

$$\begin{aligned} &\frac{\partial}{\partial \alpha_\Omega} f_\gamma(W, X + \alpha_\Omega F_\Omega) \\ &\qquad = 2\alpha_\Omega\|F_\Omega W\|_F^2 + 2\mathbf{Tr}\left(W^T X^T F_\Omega W\right), \end{aligned}$$

the step length minimizing $f_\gamma(W, X + \alpha_\Omega F_\Omega)$ is obtained as:

$$\begin{aligned} \alpha_\Omega &= -\frac{\mathbf{Tr}\left(W^T X^T F_\Omega W\right)}{\|F_\Omega W\|_F^2} = -\frac{\mathbf{Tr}\left(WW^T X_k^T F_k\right)}{\|F_\Omega W\|_F^2} \\ &= -\frac{\mathbf{Tr}\left(D_\Omega^T F_\Omega\right)}{\|F_\Omega W\|_F^2}. \end{aligned} \tag{6}$$

In the same way, the step length minimizing $f_\gamma(W + \alpha_\Phi F_\Phi, X)$ is obtained as:

$$\alpha_\Phi = -\frac{\mathbf{Tr}\left(D_\Phi F_\Phi\right)}{\|X^T F_\Phi\|_F^2}.$$

In the case of the matrix completion problem (2), Ω is defined as

$$\Omega = \left\{X \in \boldsymbol{R}^{m\times n} :\ X_{ij} = M_{ij}, \forall (i, j) \in \mathcal{I}\right\}.$$

Let us define Ω_0 as

$$\Omega_0 = \left\{ X \in \boldsymbol{R}^{m\times n} : \ X_{ij} = 0, \forall (i,j) \in \mathcal{I} \right\}.$$

Then it holds that

$$P_\Omega(X+Y) - X = P_{\Omega_0}(Y), \quad \forall X \in \Omega, \ \forall Y \in \boldsymbol{R}^{m\times n},$$

where P_{Ω_0} denotes the orthogonal projection on Ω_0. Therefore, if $X \in \Omega$, it holds that

$$F_\Omega = P_\Omega\left(X - 2D_\Omega\right) - X = -2P_{\Omega_0}\left(D_\Omega\right) \in \Omega_0,$$

and hence $X + \alpha_\Omega F_\Omega \in \Omega$. This implies that X remains in Ω in iterations if the initial value of X is in Ω. Since we can find $X \in \Omega$ easily, we assume here that $X \in \Omega$ in each iteration. Since it holds that

$$\mathbf{Tr}\left(X^T P_{\Omega_0}(X)\right) = \mathbf{Tr}\left(P_{\Omega_0}(X^T) P_{\Omega_0}(X)\right), \ \forall X \in \boldsymbol{R}^{m\times n},$$

we have that

$$\begin{aligned} \mathbf{Tr}\left(D_\Omega^T F_\Omega\right) &= -2\mathbf{Tr}\left(D_\Omega^T P_{\Omega_0}(D_\Omega)\right) \\ &= -2\mathbf{Tr}\left(P_{\Omega_0}(D_\Omega^T) P_\Omega(D_\Omega)\right) = -2\|\tilde{D}_\Omega\|_F^2, \end{aligned}$$

where $\tilde{D}_\Omega = P_{\Omega_0}(D_\Omega)$, and that $\|F_\Omega W\|_F^2 = 4\|\tilde{D}_\Omega W\|_F^2$. Then α_Ω is obtained simply as

$$\alpha_\Omega = \frac{\|\tilde{D}_\Omega\|_F^2}{2\|\tilde{D}_\Omega W\|_F^2}.$$

Therefore we obtain the update scheme of X in Algorithm 2 for the matrix completion problem as follows,

$$X + \alpha_\Omega F_\Omega = X - \frac{\|\tilde{D}_\Omega\|_F^2}{\|\tilde{D}_\Omega W\|_F^2} \tilde{D}_\Omega. \tag{7}$$

In the same way, the update scheme of W is obtained as

$$W + \alpha_\Phi F_\Phi = W - \frac{\|\tilde{D}_\Phi\|_F^2}{\|X^T \tilde{D}_\Phi\|_F^2} \tilde{D}_\Phi, \tag{8}$$

where $\tilde{D}_\Phi = P_{\Phi_0}(D_\Phi)$ and

$$\Phi_0 = \left\{ W \in \boldsymbol{R}^{n\times n} : \ W_{ii} = 0, \ \forall i \right\}.$$

These schemes reduce the computational cost for the matrix completion problem.

2.4. Parallel GPU Implementation

Algorithm 2 can be implemented on GPU and can be speed up significantly. However it does not work at all on GPU in the case of large size matrix because GPU has limited memory capacity. Because there is no inverse matrix nor singular value decomposition but only addition and multiplication of matrices in Algorithm 2, the algorithm can be parallelized easily by dividing a large matrix into several matrices. This parallel algorithm can be executed on GPU if each divided matrix size is adequate for GPU memory capacity.

Algorithm 3 Parallel NSAO-GPM for GPU

Input: $X \in \boldsymbol{R}^{pm_p \times pn_p}$, γ, η
  repeat
    $G \leftarrow \text{ptimes}(X^T, X, p, n_p)$
    $D_\Phi \leftarrow \gamma W + \text{ptimes}(G, W, p, n_p)$; $\tilde{D}_\Phi \leftarrow P_\Phi(D_\Phi)$
    $\alpha_\Phi \leftarrow \mathbf{Tr}(\text{ptimes}(\tilde{D}_\Phi^T, \tilde{D}_\Phi, p, n_p))$
    $\alpha_\Phi \leftarrow \alpha_\Phi / \mathbf{Tr}(\text{ptimes}(X^T, \tilde{D}_\Phi, p, n_p))$
    $W \leftarrow W - \alpha_\Phi \tilde{D}_\Phi$
    $G \leftarrow \text{ptimes}(W, W^T, n_p)$
    $D_\Omega \leftarrow \text{ptimes}(X, G, p, n_p)$; $\tilde{D}_\Omega \leftarrow P_\Omega(D_\Phi)$
    $\alpha_\Omega \leftarrow \mathbf{Tr}(\text{ptimes}(\tilde{D}_\Omega^T, \tilde{D}_\Omega, p, n_p))$
    $\alpha_\Omega \leftarrow \alpha_\Omega / \mathbf{Tr}(\text{ptimes}(\tilde{D}_\Omega, W, p, n_p))$
    $X \leftarrow X - \alpha_\Omega \tilde{D}_\Omega$
    $\gamma \leftarrow \gamma / \eta$
  until termination criterion is satisfied
  where
  function ptimes(U, V, p, n_p)
  parallel for $i = 0$ to $p-1$ **do**
    parallel for $j = 0$ to $p-1$ **do**
      $U_{gpu} \underset{c2g}{\leftarrow} U(in_p + 1 : (i+1)n_p, :)$
      $V_{gpu} \underset{c2g}{\leftarrow} V(:, jn_p + 1 : (j+1)n_p)$
      $G_{gpu} \leftarrow U_{gpu}^T V_{gpu}$
      $G(in_p + 1 : (i+1)n_p, jn_p + 1 : (j+1)n_p) \underset{g2c}{\leftarrow} G_{gpu}$
    end parallel for
  end parallel for
  return G
  end function
Output: low-rank solution X

Although the computing time of addition and multiplication on GPU is much shorter than CPU, GPU computing requires data transmission time between CPU and GPU. Because the execution time of addition in GPU including data transmission time is longer than CPU, this paper parallelizes only multiplications in Algorithm 2 and proposes Algorithm 3, where X is assumed to be $pm_p \times pn_p$ matrix, and each matrix multiplication is parallelized to p^2 matrix multiplication tasks. In Algorithm 3, '$\underset{c2g}{\leftarrow}$' and '$\underset{g2c}{\leftarrow}$' denote the data transmission from CPU to GPU and from GPU to CPU, respectively, and U_{gnu}, V_{gnu} and G_{gnu} are variables on GPU. The notations $G(a:b, c:d)$, $U(a:b, :)$ and $V(:, c:d)$ denote the submatrix of G formed by rows from a to b and columns from c to d, that of U formed by rows from a to b, and that of V formed by columns c to d, respectively.

Table 1. Computing time of CPU:CPU only, G0:GPU only, G1:single GPU, and G2:double GPUs [sec].

n	r	iter	CPU	G0	G1	G2
200	2	167	0.7141	0.3932	-	-
500	5	137	5.517	1.144	-	-
1000	10	129	35.42	2.778	-	-
2000	20	127	268.1	16.83	123.1	66.21
4000	40	125	2013	108.2	744.0	381.7
8000	80	126	16042	-	5254	2712
10000	100	126	30959	-	10227	5321

3. NUMERICAL EXAMPLES

This section gives numerical examples of the matrix completion problem using CPU, single GPU and double GPUs. All numerical experiments were run in MATLAB 2010b on a PC with an Intel Core i7 3.4GHz CPU and 16GB of RAM. The experiments of CPU computing utilize 4 cores using MATLAB command `maxNumCompThreads(4)`. The experiments of GPU computing utilize NVIDIA GeForce GTX 580 with 3GB of RAM and Jacket for MATLAB, which is a software for GPU computing in MATLAB.

For each experiment, the optimal solution $X_{opt} \in \boldsymbol{R}^{n\times n}$ of rank r is generated as YY^T, where $Y \in \boldsymbol{R}^{n\times r}$ is generated using i.i.d. gaussian entries, and X_{opt} is normalized so that its maximum singular value is 1. The index set $\mathcal{I}$ is generated using bernoulli $\{0,1\}$ random variables with a mean support size of q where q/n^2 is the bernoulli probability for an index (i,j) to belong to $\mathcal{I}$. We use $q = 0.12$ in all experiments. In NSAO-GPM, $\eta = 1.1$, the initial value of γ is 1×10^{-2}, and that of X is generated as $X_{ij} = M_{ij}$ if $(i,j) \in \mathcal{I}$ and $X_{ij} = 0$ if $(i,j) \notin \mathcal{I}$.

We solved randomly created matrix completion problems for each set of (n,r) using NSAO-GPM with update schemes (7) and (8) for CPU and GPU, where there is no transmission between CPU and GPU, and parallel NSAO-GPM for single GPU and double GPUs. In parallel NSAO-GPM, we utilized $m_p = n_p = 2000$. Both algorithms iterate until they achieve $\|X_{opt} - X_k\|_F / \|X_k\|_F < 10^{-3}$. Table 1 shows the results, where 'iter' denotes the number of iterations, 'CPU' and 'G0' denote the results of NSAO-GPM using CPU and GPU, and 'G1' and 'G2' denote the result of parallel NSAO-GPM using single GPU and double GPUs, respectively. Due to GPU memory capacity, NSAO-GPM cannot be applied to the problem of $n > 4000$ in GPU computing (G0).

We can see that NSAO-GPM recovers low-rank matrices and can be speed up significantly by GPU computing if the matrix size is adequate for GPU memory capacity. We can also see that parallel NSAO-GPM is faster than non-parallel algorithm using CPU computing and that the computing time is reduced almost linearly according to the number of GPUs.

4. CONCLUSION

This paper proposes the parallel NSAO-GPM algorithm for the matrix rank minimization problem. Numerical examples show that the parallel GPU algorithm is faster than non-parallel algorithm via CPU and speed up almost linearly according to the number of GPUs. The advantage of NSAO-GPM is that it can be parallelized and be speed up almost linearly because it requires only addition and multiplication but no inverse matrix nor singular value decomposition. This implies that parallel NSAO-GPM can run on many parallel GPUs and a many-core CPU and will be applied to much larger problems in the near future.

5. REFERENCES

[1] K. Mohan and M. Fazel, "Reweighted nuclear norm minimization with application to system identification," *Proc. American Control Conference*, pp. 2953 – 2959, 2010.

[2] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational Mathematics*, vol. 9, pp. 717–772, 2009.

[3] T. Takahashi, K. Konishi, and T. Furukawa, "Reweighted l2 norm minimization approach to image inpainting based on rank minimization," *Proc. of IEEE International Midwest Symposium on Circuits and System*, 2011.

[4] T. Ding, M. Sznaier, and O.I. Camps, "A rank minimization approach to video inpainting," *Proc. of IEEE International Conference on Computer Vision*, pp. 1–8, 2007.

[5] K. Mohan and M. Fazel, "Iterative reweighted least squares for matrix rank minimization," *Proc. of the Allerton Conference on Communications, Control, and Computing*, pp. 653–661, 2010.

[6] J. P. Haldar and D. Hernando, "Rank-constrained solutions to linear matrix equations using powerfactorization," *IEEE Signal Process. Lett.*, vol. 16, no. 7, pp. 584–587, 2009.

[7] J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM J. Optimiz.*, , no. 4, pp. 1956–1982, 2010.

[8] S. Ma, D. Goldfarb, and L. Chen, "Fixed point and bregman iterative methods for matrix rank minimization," *Mathematical Programming*, vol. 128, no. 1-2, pp. 321–353, 2011.