

Appendix to “Large-scale Structured Sparsity via Parallel Fused
Lasso on Multiple GPUs” submitted to the Journal of
Computational and Graphical Statistics

Taehoon Lee, Joong-Ho Won, Johan Lim, and Sungroh Yoon

January 5, 2017

1 MM algorithm for 2D FLR

Yu et al. (2015) propose an MM algorithm to solve (1). The MM algorithm iterates between two steps: majorization and minimization. Given the current estimate $\widehat{\beta}^0$ of the optimal solution to (1), the majorization step constructs a majorizing function $g(\beta|\widehat{\beta}^0)$ such that $f(\widehat{\beta}^0) = g(\widehat{\beta}^0|\widehat{\beta}^0)$ and $f(\beta) \leq g(\beta|\widehat{\beta}^0)$ for all $\beta \neq \widehat{\beta}^0$. The minimization step updates the estimate with $\widehat{\beta} = \arg \min_{\beta} g(\beta|\widehat{\beta}^0)$. Motivated by Hunter and Li (2005), Yu et al. (2015) proposed to perturb (1) slightly and successively minimize this by MM. The ϵ -perturbed version of (1) is given by

$$f_{\epsilon}(\beta) = \frac{1}{2}\|y - X\beta\|_2^2 + \lambda_1 \sum_{j=1}^p \left\{ |\beta_j| - \epsilon \log \left(1 + \frac{|\beta_j|}{\epsilon} \right) \right\} + \lambda_2 \sum_{(j,k) \in E} \left\{ |\beta_j - \beta_k| - \epsilon \log \left(1 + \frac{|\beta_j - \beta_k|}{\epsilon} \right) \right\}, \quad (\text{A1})$$

and its majorizing function $g_{\epsilon}(\beta|\widehat{\beta}^0)$ is

$$g_{\epsilon}(\beta|\widehat{\beta}^0) = \frac{1}{2}\|y - X\beta\|_2^2 + \lambda_1 \sum_{j=1}^p \left\{ |\widehat{\beta}_j^0| - \epsilon \log \left(1 + \frac{|\widehat{\beta}_j^0|}{\epsilon} \right) + \frac{\beta_j^2 - (\widehat{\beta}_j^0)^2}{2(|\widehat{\beta}_j^0| + \epsilon)} \right\} + \lambda_2 \sum_{(j,k) \in E} \left\{ |\widehat{\beta}_j^0 - \widehat{\beta}_k^0| - \epsilon \log \left(1 + \frac{|\widehat{\beta}_j^0 - \widehat{\beta}_k^0|}{\epsilon} \right) + \frac{(\beta_j - \beta_k)^2 - (\widehat{\beta}_j^0 - \widehat{\beta}_k^0)^2}{2(|\widehat{\beta}_j^0 - \widehat{\beta}_k^0| + \epsilon)} \right\}. \quad (\text{A2})$$

For the estimate coefficient vector $\widehat{\beta}^{(r)}$ at the r th iteration, the majorizing function $g_{\epsilon}(\beta|\widehat{\beta}^{(r)})$ is minimized over β when $\partial g_{\epsilon}(\beta|\widehat{\beta}^{(r)})/\partial\beta = 0$. Otherwise, to solve a linear system of equations

$$(X^T X + \lambda_1 A^{(r)} + \lambda_2 B^{(r)})\beta = X^T y, \quad (\text{A3})$$

where $A^{(r)} = \text{diag}(a_1^{(r)}, a_2^{(r)}, \dots, a_p^{(r)})$ with $a_j^{(r)} = 1/(|\widehat{\beta}_j^{(r)}| + \epsilon)$ for $1 \leq j \leq p$, and $B^{(r)} = (b_{jk}^{(r)})_{1 \leq j, k \leq p}$ is a symmetric and positive semidefinite matrix with

$$b_{jj}^{(r)} = \sum_{k:(j,k) \in E} \frac{1}{|\widehat{\beta}_j^{(r)} - \widehat{\beta}_k^{(r)}| + \epsilon}, \quad j = 1, \dots, p, \\ b_{jk}^{(r)} = b_{kj}^{(r)} = -\frac{1}{|\widehat{\beta}_j^{(r)} - \widehat{\beta}_k^{(r)}| + \epsilon}, \quad \forall (j, k) \in E.$$

A standard procedure to solve (A3) is the Cholesky decomposition, followed by the backward substitution, both of which are difficult to parallelize. Yu et al. (2015) observed that the matrix term

$$M^{(r)} = \lambda_1 A^{(r)} + \lambda_2 B^{(r)}$$

in (A3) can be constructed in $O(m)$ operations, which has an advantage over the other procedures for solving (1), and considered a preconditioned conjugate gradient (PCG) method that uses this term as the preconditioner. For the standard FLR, the preconditioner solve is a tridiagonal linear system, and can be conducted in parallel by using a variant of the cyclic reduction algorithm (Stone, 1973) for GPU (Zhang et al., 2010). It is observed that for the standard FLR, the MM algorithm tends to converge within a few tens of iterations for tens of thousands of p (Yu et al., 2015).

However, for the 2D FLR, each preconditioner solve involves a *block* tridiagonal system, and the strategy for the standard FLR appears not to be effective any more. Specifically, for a p_1 -by- p_2 grid, we have

$$M^{(r)} = \begin{bmatrix} T_1 & D_1 & & & & \\ D_1 & T_2 & D_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & D_{p_2-2} & T_{p_2-1} & D_{p_2-1} \\ & & & & D_{p_2-1} & T_{p_2} \end{bmatrix} \quad (\text{A4})$$

where T_k is a $p_1 \times p_1$ tridiagonal matrix, and D_k is a $p_1 \times p_1$ diagonal matrix. Inverting (A4) amounts to solve

$$D_{j-1}x_{j-1} + T_j x_j + D_j x_{j+1} = v_j, \quad j = 1, \dots, p_2, \quad D_0 = D_{p_2} = 0. \quad (\text{A5})$$

Block cyclic reduction eliminates x_{2k-1} and x_{2k+1} by multiplying $D_{2k-1}(T_{2k-1})^{-1}$ to the $2k-1$ st equation and $D_{2j}(T_{2j+1})^{-1}$ to the $2k+1$ st equation, and by subtracting them from the $2k$ th equation (Gander and Golub, 1997). Each reduction step involves three p_1 -by- p_1 tridiagonal system solves. Ultimately, block cyclic reduction requires $3p_2$ tridiagonal solves. Unless p_1 is small, this strategy is not very effective.

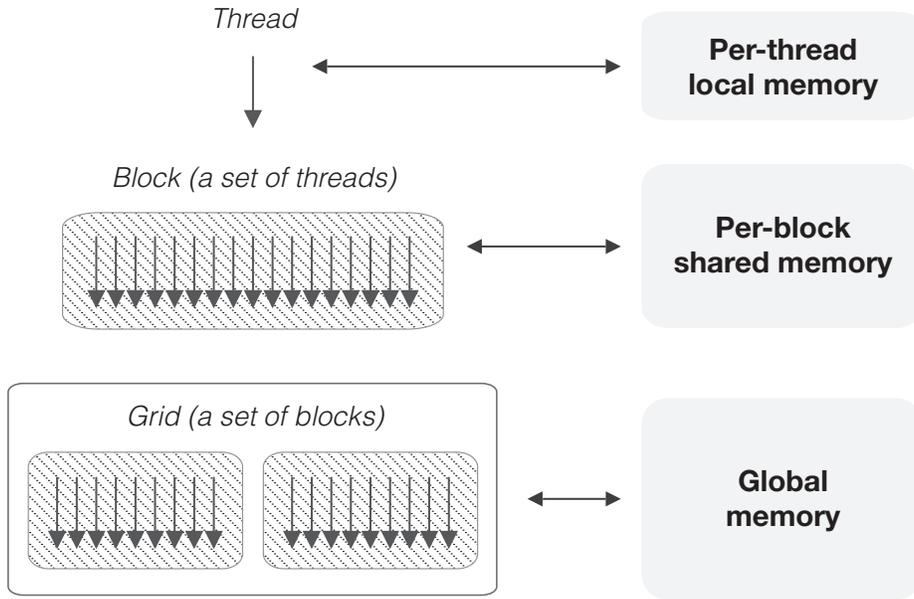


Figure A1: Memory hierarchy of the CUDA (Owens et al., 2008).

2 Overview of the GPU architecture

In this section, we focus primarily on NVidia’s CUDA GPUs, although many of the principles can also be applied to other single-instruction-multiple-data (SIMD) architectures. A CUDA GPU consists of thousands of cores that can execute the same program, called a *thread*, in parallel. A certain number of cores (depending on the “compute capability” of the GPU) are contained in a streaming multiprocessor (SM). A GPU is a scalable array of SMs. The group of threads that executes the same program simultaneously on the same SM is called a *warp*. The global memory of a GPU is accessible from all SMs; each SM is equipped with a small amount of shared memory with which the threads of a warp can communicate with each other (NVidia Corporation, 2015). A thread can be assigned registers from the register file of the SM. From a programming point of view, the programmer allocates memory on the GPU, copies the data from the host’s main memory to the GPU’s global memory, and specifies a program called a *kernel* whose multiple instances run on the SM. After the kernel threads terminate, the programmer copies the data back to the host. The memory hierarchy of CUDA is illustrated in Fig. A1.

For the 2D FLR, the split Bregman algorithm consists mostly of scalar-vector multiplication, vector-vector addition, matrix-vector multiplication, and squared Euclidean norm

calculation. These operations have a typical SIMD structure, and can be parallelized easily using the cuBLAS. The FFT used in solving the linear system can also be performed in parallel using the cuFFT library. Both cuBLAS and cuFFT are parts of the CUDA toolkit version 7.5 (NVIDIA Corporation, 2012) Xin et al. (2014) Zhu (2016).

3 Image denoising examples



β (original)

y (corrupted)

$\hat{\beta}$ (denoised)

Figure A2: Image denoising with Peppers, Cameraman, and Baboon.

References

Gander, W. and G. H. Golub (1997). Cyclic reduction – history and applications. *Scientific Computing (Hong Kong, 1997)*.

- Hunter, D. R. and R. Li (2005). Variable selection using mm algorithm. *The Annals of Statistics* 33(4), 1617–1642.
- NVidia Corporation (2012). NVidia CUDA Toolkit. <https://developer.nvidia.com/cuda-toolkit>. [Online; accessed 13-April-2016].
- NVidia Corporation (2015). NVidia CUDA C Programming Guide. <http://docs.nvidia.com/cuda/cuda-c-programming-guide>. [Online; accessed 13-April-2016].
- Owens, J. D., M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips (2008). GPU computing. In *Proceedings of the IEEE*, Volume 96, pp. 879–899.
- Stone, H. S. (1973). An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *Journal of the ACM (JACM)* 20(1), 27–38.
- Xin, B., Y. Kawahara, Y. Wang, and W. Gao (2014). Efficient generalized fused lasso with its application to the diagnosis of alzheimers disease. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.
- Yu, D., J.-H. Won, T. Lee, J. Lim, and S. Yoon (2015). High-dimensional fused lasso regression using majorization–minimization and parallel processing. *Journal of Computational and Graphical Statistics* 24(1), 121–153.
- Zhang, Y., J. Cohen, and J. D. Owens (2010). Fast tridiagonal solvers on the GPU. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 127–136. ACM.
- Zhu, Y. (2016+). An augmented ADMM algorithm with application to the generalized lasso problem. *Journal of Computational and Graphical Statistics*, to appear.