A Rcpp-CUDA framework for Parallel Linear Model Learning(RcppGLM)

Xin Zhou(ECE)

Abstract

R project is a mature programming-language for statistical computing and graphics. With the area of statistical learning are experiencing rapid growth in size of data, a lot of parallel computing approaches for R has been developed, including computing with R on clusters, multi-core system and grid computing. In this project, we implement a CUDA version of Logistic Regression and apply it to process the MNIST digits handwriting classification problem. As the result, we find that CUDA parallel framework embeded in R environment provide 8x speed up comparing with build-in R functions such as %*%. which are used as baseline performance of multiple class classification problem of Logistic Regression.

Background

- **Rmpi** The package Rmpi is an wrapper to MPI, providing an R interface to low-level MPI functions.
- **R/parallel** The package rparallel (Vera, Jansen, and Suppi 2008) enables automatic parallelization of loops without data dependencies by adding a signle function: **runParallel()**. This package uses master-slave architecture, its implementation is based on C++, and combines low level operating system calls to manage processes, threads and inter-process communications.
- **Multicore** This package provides functions of parallel execution(such as **mclapply**()) on machines with multi-processors.
- **gputools** R+GPU provides several basic linear algebra functions with R interfaces that execute with different degree of parallism on GPU.
- **CudaBayesreg** cudaBayesreg (Silva 2011) is a package which implements in CUDA the core of a Bayesian multilevel model for the analysis of brain fMRI data. Given that it is developed for Bayes regression model, cudaBayesreg provide parallelized version of Gibbs sample and MCMC.
- **RCUDA** RCUDA package is similar in nature to OpenCL, and it is used to provide entire SDK of CUDA to R users.

Therefore, R community has developed several state-of-art packages which executing parallel computing on R, in addition, there is no an well developed CUDA frame interface to execute all kinds of CUDA snippets on R via inline mode or .Call mode like the Rcpp package (Eddelbuettel and Francois 2011). Therefore, my objective of this project, besides completing a project itself, I also wish to generate a generalized CUDA framework for R programming-language.

Machine Learning and CUDA

Deep neural networks have been successful at solving most kind of tasks such as motif recognition and natural language processing, In addition, GPUs have played a significant role in the practical implementation of deep neural networks.

Ax the result, In 2014, cuDNN(Chetlur et al. 2014) was published, since Sharan Chetlur et.al have provided a library of efficient implementations of deep learning primitives. cuDNN has provided a flexible and easy-to-use C API for deep learning and convert the cNN into a special form of batched convolution which is suitable for GPU's architecture.

Rcpp package and R Tool-Chain

Rcpp package(Eddelbuettel and Francois 2011) provides C++ classes that greatly facilitate interfacing C or C++ code in R packages using the **.Call**() interface provided by R and provides matching C++ classes for large number of basic R data type. Therefore, this package is very useful to programmer create a Rcpp version of C or C++ library. So far, there has a lot of high performance c library has been transplanted to R by Rcpp, for instance, RcppArmadillo, RcppEigen, RInside etc.

Also, there has exsited a C-API easy Rcpp deep learning package **RcppDL** based on Bengio's paper(Bengio et al. 2007), however, it just a very tiny version and executes serially on CPU, therefore, what we wish to do is to apply CUDA programming on these deep learning algorithm.

Application: Rcpp can accelerate Algebra Operation

To compare the progress, I created a new R package `RcppGLM`, with the help of this package, what I wish to do is not only execute my stochastic average gradient algorithm for Deep learning or Generalize Linear Regression(GLM). All of these algorithms have a optimization steps and we all can rewrite their optimization as a finite summed function.

Secondly, to create this package and apply parallel algorithm in `R project`, I used the Rcpp as the backend of my R package.

But since this method need CUDA and Openmp as the backend. I have to define a new `Makevars` file, which is specific to Rcpp as a `Makefile`. And by now, I have two part of my job. One is build my `RcppGLM` based on Openmp. Another is to modify my algorithm `SAG`. And in my SAG, the matrix multiplication is necessary. Therefore, I have rewritten the parallelized DEGMM algorithm and Stressan algorithm for my `RcppGLM` package. And on the 2-core i5 processors, I have achieved 2.5x speedup so far on 4 threads by openmp.

parallel	196.108	0.560	51.931
series	127.376	0.568	128.422

Given that matrix operations such as matrix multiplication or matrix inversion are building block of machine learning, if we can accelerate basic algbra operations by embedding CUDA code in R environment by Rcpp, I think we also can create an fast framework for machine by switch our machine learning code's backend.

Description of Logistic Regression Algorithm

Multiple Logistical Regression

- Define $\mathbf{Z}$ as a vector and $Z_i \in \{0, 1\}$.
- N total number of population
- each population's observation is n_i; $\sum_{i=1}^{N} n_i = M$
- $\mathbf{Y}$ has N population(N rows). If observation has 2 categories, then regrade to binomial distribution and $\mathbf{Y}$ is a column vector. If $J \geq 3$ categories, $\mathbf{Y}$ will have $J-1$ columns.
- Since we have J classes.
- So, we have $Y(N \times (J-1)) = X(N \times p)\beta(p \times (J-1))$.
- The β coefficient has $J-1$ columns

$$\ln(\frac{\pi_{ij}}{\pi_{iJ}}) = X\beta_j$$

- Here β_j means the j th column of matrix β.
- Since $\sum_{j=1}^{J} \pi_{ij} = 1$, which means that for each response Y_i, the summed probability in predictors space is 1.

$$\pi_{ij} = \frac{e^{X_i\beta_j}}{1+\sum_{j=1}^{J-1} e^{X_i\beta_j}} \qquad j \neq J \pi_{iJ} = \frac{1}{1+\sum_{j=1}^{J-1} e^{X_i\beta_j}} \qquad j = J$$

- In Logistical regression, since we have N obvervation $\mathbf{Y}$, and we use $\mathbf{Y}_i$ represents each observation $(0, 0, 0, ..1, ..0)$.
- So the liklihood function is

$$L(\beta) = -\log(\prod_{i=1}^{N}\prod_{j=1}^{J} \pi_{ij}^{\mathbf{Y}_{ij}})$$

$$L(\beta) = -\sum_{i=1}^{N}\sum_{j=1}^{J} \mathbf{Y}_{ij}\log[\pi_{ij}]$$

$$= -\sum_{i=1}^{N}[\sum_{j=1}^{J-1} \mathbf{Y}_{ij} X_i\beta_j - n_i(\log[1+\sum_{j=1}^{J-1} e^{X_i\beta_j}]]$$

$$= -\sum_{i=1}^{N}[\sum_{j=1}^{J-1} \mathbf{Y}_{ij} \sum_{k=1}^{p} X_{ik}\beta_{kj} - n_i(\log[1+\sum_{j=1}^{J-1} e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}}]]$$

P.S:

$$n_i = \sum_{j=1}^{J} Y_{ij} == 1; \qquad Y_{ij} \in \{0, 1\}$$

- Do line search :

$$\frac{\partial L(\beta)}{\partial \beta_{kj}} = -\sum_{i=1}^{N}[\mathbf{Y}_{ij}X_{ik} - \frac{n_i X_{ik} e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}}}{1+\sum_{j=1}^{J-1} e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}}}]$$

$$= -\sum_{i=1}^{N}[\mathbf{Y}_{ij}X_{ik} - n_i X_{ik}\pi_{ij}] = -X^T(Y - \Pi)$$

- Therefore, I think we can build the $\nabla = [\nabla_{kj} = \sum_{i=1}^{N}[\mathbf{Y}_{ij}X_{ik} - n_i X_{ik}\pi_{ij}]]$.
- Do LBGFS:

$$\textbf{Hessian} = \frac{\partial^2 L(\beta)}{\partial\beta_{kj}\partial\beta_{k'j'}} =$$

$$\sum_{i=1}^{N} n_i X_{ik} e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}} \frac{X_{ik'} e^{\sum_{k=1}^{p} X_{ik}\beta_{kj'}}}{(1+\sum_{j=1}^{J-1} e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}})^2} = -\sum_{i=1}^{N} n_i X_{ik} X_{ik'} \pi_{ij}\pi_{ij'} \qquad j \neq j'$$

$$\sum_{i=1}^{N} n_i X_{ik} \frac{(1+\sum_{j=1}^{J-1} e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}}) X_{ik'} e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}} - e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}} X_{ik'} e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}}}{(1+\sum_{j=1}^{J-1} e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}})^2} =$$

$$n_i X_{ik} X_{ik'} \frac{e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}}}{1+\sum_{j=1}^{J-1} e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}}} \left(1 - \frac{e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}}}{1+\sum_{j=1}^{J-1} e^{\sum_{k=1}^{p} X_{ik}\beta_{kj}}}\right) =$$

$$\sum_{i=1}^{N} n_i X_{ik} X_{ik'} \pi_{ij}(1-\pi_{ij}) \qquad j = j'$$

- Therefore, we can derivate that $\mathbf{H}_{ij} = \sum_{i=1}^{N} n_i X_{ik} X_{ik'} \pi_{ij} \pi_{ij'}^{(1-\mathbf{sgn}(j,j'))} (1-\pi_{ij'})^{\mathbf{sgn}(j,j')}$

$$\begin{aligned} \mathbf{sgn}(x,y) &= 1 \qquad x = y \\ \mathbf{sgn}(x,y) &= 0 \qquad x \neq y \end{aligned}$$

- In Newton Rasphon Algorithm, we need that

$$\beta^{(t+1)} = \beta^{(t)} - \alpha(t)\mathbf{H}^{-1}\nabla$$

- In linear algebra, for each column of β we can represent that update β_j

$$\sum_{i=1}^{N} [\mathbf{Y}_{ij} X_{ik} - n_i X_{ik} \pi_{ij}] = \sum_{i=1}^{N} [\mathbf{Y} - \mathbf{U}]_{ij} X_{ik} \;\; X^T(\mathbf{Y} - \mathbf{U}) = \nabla_{p\times(J-1)}$$

Stochastic average gradient algorithm

Comapring with the full gradient, when we process data whose observation N is very large, we always apply stochastic average gradient to reduce algorithm's time consuming in each iteration.

In large-scale machine learning, there are plethora of optimization problems based on empirical risk minization principle in statistical learning theory, this class of problems involve computing a minimizer of a finite sum of a finite set of smooth functions, since the sum structure is a natual form of loss function over large numbers of data points.

Additionally the most widly successful class of algorithms to solve sum structure problems are stochastic gradient methods. While stocahstic average method is a faster algorithm which achieves $\mathcal{O}(\rho^k)$ coverage rate when summed function is strongly-convex(Roux, Schmidt, and Bach 2012, schmidt2013minimizing), it reduces the cost of iteration of gradient descent by keeping last iteration's gradient in memory.

$$g(x) := \frac{1}{n}\sum_{i=1}^{n} f_i(x) \tag{1}$$

SAG applied to logistic regression

We will implement the SAG algorithm, which will contain a solver for L-2 norm regularized logistic regression. For multiple data logistic regression, its loss function can be written as (2).

$$\frac{\lambda||x||^2}{2} + \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp(-b_i a_i^T x)\right) \tag{2}$$

In (2), a_i is predictor in logistic regression problem and $a_i \in \mathcal{R}^p$, while b_i is response of this problem, since logistic regression is for categorical problem, $b_i \in \{-1, 1\}$. Besides, x is the coefficient we will estimate and regularize parameter λ is to control our fitting parameters.

If we treat $f_i(x) = \log\left(1 + \exp(-b_i a_i^T x)\right) + \frac{\lambda||x||^2}{2}$, L-2 regularize logistic regression problem is a optimization problem for finite sum and SAG can be applied to solving this class of problems.

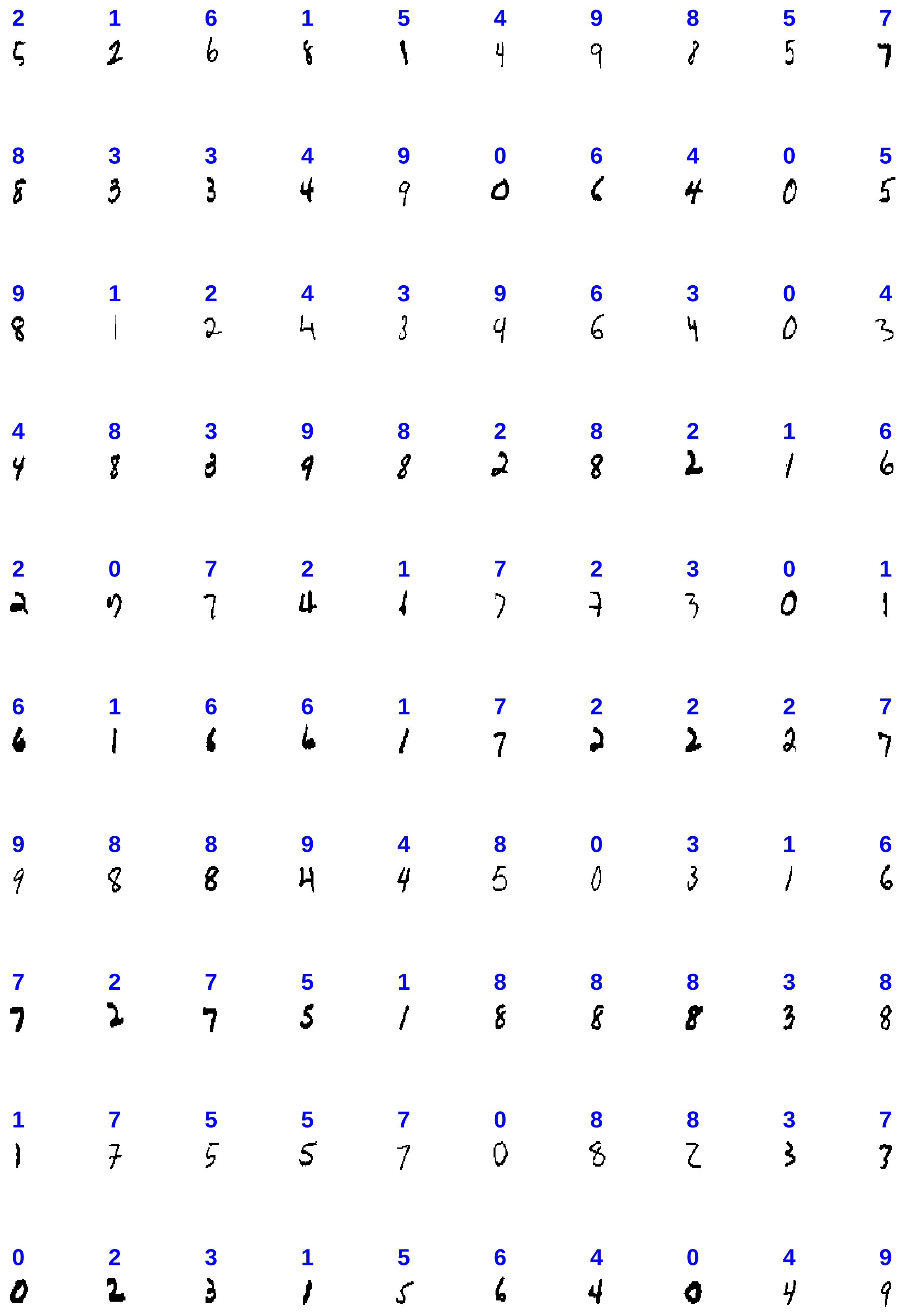
$$\begin{gathered} \min_{x \in \mathcal{R}^p} \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp(-b_i a_i^T x)\right) + \frac{\lambda||x||^2}{2} \\ x^{t+1} = x^t - \frac{\alpha}{n}\sum_{i=1}^{n} s_i^t \end{gathered} \tag{3}$$

$$s_i^t = \begin{cases} \nabla(\log\left(1 + \exp(-b_i a_i^T x)\right) + \frac{\lambda||x||^2}{2}) & \text{for randomly choose } i, \\ \\ s_i^{t-1} & \text{otherwise} \end{cases}$$

Implementation of RcppGLM

Results

To be more clear, we apply our Logistic Regression Classifier to recognize the MNIST handwriting digits. To realize this target, I have design a new **ggplot** object to combine all 28×28 mnist digit picture with its predicted title.


2 1 6 1 5 4 9 8 5 7
8 3 3 4 9 0 6 4 0 5
9 1 2 4 3 9 6 3 0 4
4 8 3 9 8 2 8 2 1 6
2 0 7 2 1 7 2 3 0 1
6 1 6 6 1 7 2 2 2 7
9 8 8 9 4 8 0 3 1 6
7 2 7 5 1 8 8 8 3 8
1 7 5 5 7 0 8 8 3 7
0 2 3 1 5 6 4 0 4 9

In our **RcppGLM** Package, we have design R, C++ and CUDA version Logistical Regression Classifier, also, you can recheck the algorithm in section 2.

Parallel Chunks

Logistical Regression Problem is a typical Optimization Problem. Therefore, we can split the algorithm into general three parts:

- **objval** : Calculating the Objective loss function
- **gradient** : Calculating gradient for gradient descent
- **linesearch** : Finding out proper step size for gradient descent

When the observation number N is larger, we can use **reduce model** for the summation of Log-liklihood. For instance, we also can use **CUBLAS::cublasDasum** function to complete this reduce sum on GPU.

In addition, When we calculate the gradient of loss function, actually, we will process a matrix multiplication. Therefore, we use a block matrix multiplication to complete the gradient calculation. CUBLAS also provide a similar function **CUBLAS::cublasDgeam**

Benchmark

Based on our parallel scheme, we compare the performance of my Logistic Regression Algorithm between R version and CUDA version.

In this benchmark experiment, our R and C version code is running on the CPU Intel Core 3720QM(2.6 GHz) and the CUDA code is running on NVIDIA GeForce GT 650M with CUDA toolkit 5.5, and CUDA computation Capability is 3.0.

GT 650M has 384 CUDA cores and it can provide as more as $1024 \times 1024 \times 64$ threads.

Our data is the 2000 MNIST digits and initialized coefficients w of logistic regression model are all 0.

Convergence and Classification Error

When the three model converged, their δ are same.

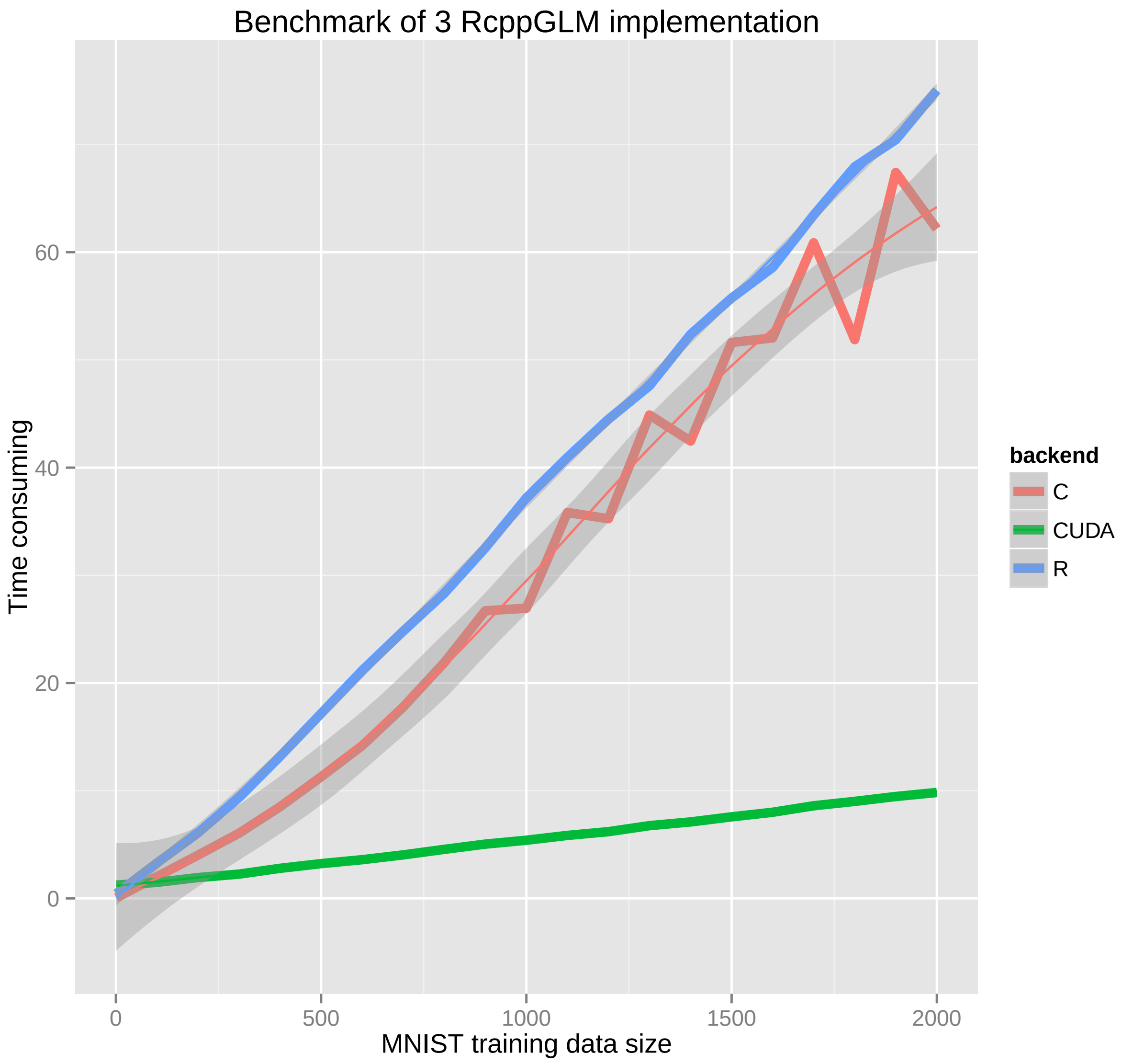
R	C	CUDA
1e-06	1e-06	1e-06

In addition, the misclassification error on test dataset is

R	C	CUDA
0.155	0.155	0.155

Speed comparison

To test the speedup of parallel logistic regression, we sampled different size of training data for three version LR. As the result, we find that, the CUDA version can achieve as much as 7.631852x speed up. And also we found that CUDA version's RcppGLM's speedup increases when our training sample increases.



.

Future

Although, we have built a Parallel Framework for Logistic Regression Model. We only write the Full Gradient Version Logistical Regression on GPU and get about 8x speed up. Therefore, I assume that if we apply the Stochastic Average Gradient scheme, which is proper for multi-thread computation, we will get much more speed-up.

References

Bengio, Yoshua, Pascal Lamblin, Dan Popovici, Hugo Larochelle, and others. 2007. "Greedy Layer-Wise Training of Deep Networks." *Advances in Neural Information Processing Systems* 19: 153.

Chetlur, Sharan, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. "cudnn: Efficient Primitives for Deep Learning." *arXiv Preprint ArXiv:1410.0759*.

Eddelbuettel, Dirk, and Romain Francois. 2011. "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software* 40 (8) (April 13): 1–18. http://www.jstatsoft.org/v40/i08.

Roux, Nicolas L, Mark Schmidt, and Francis R Bach. 2012. "A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets." *Advances in Neural Information Processing Systems*: 2663–2671.

Silva, AR Ferreira da. 2011. "cudaBayesreg: parallel Implementation of a Bayesian Multilevel Model for FMRI Data Analysis." *Journal of Statistical Software* 44 (4): 1–24.

Vera, Gonzalo, Ritsert C Jansen, and Remo L Suppi. 2008. "R/Parallel–Speeding up Bioinformatics Analysis with R." *BMC Bioinformatics* 9 (1): 390.