Design Document for Parallelization R Package glmm

Sydney Benson

July 30, 2018

Abstract

This design document will give an overview of the changes made to the R package `glmm` with respect to parallel computing. We use parallel computing in this package to increase the speed of the calculation of the value of the log-likelihood approximation, gradient and hessian for use with MCLA.

1 Introduction

This project is meant to increase the overall speed of computation for the `glmm` R package using a parallel computing structure. The parallel computing structure will be responsible for producing the value of the log-likelihood approximation, gradient and hessian for each section of a divided u matrix. We'll start this design document by discussing the various packages being used for this project. Then, the parallel computing process is given; the cluster of processing cores is created, required packages and variables are added to the cluster, the u matrix is separated and processed and the returned values are recombined to create the true value of the log-likelihood approximation, gradient and hessian for the full u matrix.

2 The Packages

2.1 `parallel`

The first package we use for this parallelization is the R package `parallel`. This package allows for the detection of the number of cores in a computing device and the creation of clusters which allows us to access all of the available cores in the device.

2.2 doParallel

`doParallel` is utilized in the parallelization process to allow the use of the `foreach` function within the parallel package. This package allows the function to use the cluster created by `parallel`.

2.3 itertools

`itertools` is used in the parallelization process to divide the matrix into even parts, with the number of parts depending on the number of cores in the cluster.

3 The Process

The parallelization of the function will be completed in two steps. First, the value of the log-likelihood approximation and the gradient will be calculated in one parallelized function. Then, this information will be used to calculate the hessian in a second, separate, parallelized function. Therefore, the cluster will need to be prepared and closed, and the matrix will need to be divided two separate times.

3.1 Preparing the Cluster

3.1.1 detectCores

We begin by using the `detectCores` command in order to find the number of cores available in the computing device being used. The actual number of cores we will use for calculations is the number of cores in the device minus one. This enables the user to continue using the device while the function is being computed.

3.1.2 makeCluster

After detecting how many cores we have available for our cluster, we can create a cluster of our cores using `makeCluster` where the only argument is the number of cores available for use.

3.1.3 registerDoParallel

Now that we have our cluster made, we can register the cluster for use with the `foreach` function.

3.1.4 `clusterEvalQ`

The `clusterEvalQ` function allows us to download any necessary packages to work within our cluster. The package that will need to be downloaded within our cluster is `itertools`.

3.1.5 `clusterExport`

The final function we need to use to set up our cluster is the `clusterExport` function. This function allows us to bring any variable from the global environment into the cluster environment for use there. We will need to use this to bring all necessary variables for the `.C` function into the cluster environment.

3.2 Separating the Matrix

Next, we separate our calculations between our available cores. This separation will be done using the `isplitRows` function. Using this function, we will split our u matrix row-wise into parts, the number of parts being determined by the number of cores in our cluster.

3.3 Calculating the Value, Hessian and Gradient

To calculate the value and gradient, we have to send each of our chunks of the u matrix through the `.C` function, which allows us to run the objective function `C` function. We can do this using the `foreach` function and the `%dopar%` operator. We then have the value and gradient values from each chunk of the u matrix returned to us as part of a list.

3.3.1 The Value

We begin by thinking about combining the values given by each core in the simple case, not accounting for any overflow issues or uneven division of the matrix among the cores. We can find the value for the entire u matrix using

$$l_m(\theta|y) = \log\left(\frac{1}{r}\sum_{i=1}^{r} e^{v_i}\right) \tag{1}$$

where r is the number of cores being used and v_i is the value obtained from each core. However, we might face the issue of an uneven distribution of the u matrix among the cores because the number of rows in the u matrix is not divisible by the number of cores being utilized. In this case, we need to alter equation 1 to be

$$l_m(\theta|y) = \log\left(\frac{1}{m}\sum_{i=1}^{r} j_i \; e^{v_i}\right) \tag{2}$$

where m is the number of rows of the u matrix, j_i is the number of rows of the chunk of the u matrix being processed by the ith core. Finally, we might run into an overflow problem. This occurs when a number becomes so great that it is stored as infinity by `R`. Thus, the equation changes to

$$l_m(\theta|y) = a + \log\left(\frac{1}{m}\sum_{i=1}^{r} j_i \; e^{v_i - a}\right) \tag{3}$$

where $a = \max v_i$.

3.3.2 The Gradient

Recombining the gradient from each core requires slightly more work than the recombination of the values. We'll understand the complexity of this equation using the simplest case possible, one with no computational instability and an even distribution of the portions of the u matrix among the cores. First, the gradient we get from processing the u matrix on a single core follows this equation:

$$\nabla l_m(\theta|y) = \frac{\sum_{k=1}^{m} \left[\nabla \log f_\theta(u_k, y)\right] \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)}}{\sum_{k=1}^{m} \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)}} \tag{4}$$

where m is the number of rows in the u matrix. This means that the gradient received from each core via parallel processing will follow this equation. Thus, we face the issue of having an incorrect denominator if we simply add the gradients from each core. To fix this, we need to first multiply the gradient from each core by the denominator of that gradient. We can obtain $\sum_{k=1}^{m} \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)}$ since we know that $l_m(\theta|y) = \log\left(\frac{1}{m}\sum_{k=1}^{m} \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)}\right)$ and $l_m(\theta|y) = \log\left(\frac{1}{r}\sum_{i=1}^{r} e^{v_i}\right)$. Thus,

$$\log\left(\frac{1}{m}\sum_{k=1}^{m}\frac{f_\theta(u_k,y)}{\tilde{f}(u_k)}\right)=\log\left(\frac{1}{r}\sum_{i=1}^{r}e^{v_i}\right) \tag{5}$$

$$\frac{1}{m}\sum_{k=1}^{m}\frac{f_\theta(u_k,y)}{\tilde{f}(u_k)}=\frac{1}{r}\sum_{i=1}^{r}e^{v_i} \tag{6}$$

$$\sum_{k=1}^{m}\frac{f_\theta(u_k,y)}{\tilde{f}(u_k)}=\frac{m}{r}\sum_{i=1}^{r}e^{v_i} \tag{7}$$

therefore, we can combine each the gradient from each core using

$$\nabla l_m(\theta|y)=\frac{\frac{m}{r}\sum\limits_{i=1}^{r}e^{v_i}\ g_{i,k}}{\frac{m}{r}\sum\limits_{i=1}^{r}e^{v_i}} \tag{8}$$

$$\nabla l_m(\theta|y)=\frac{\sum\limits_{i=1}^{r}e^{v_i}\ g_{i,k}}{\sum\limits_{i=1}^{r}e^{v_i}} \tag{9}$$

where $k=1,2,\ldots,s$, s is the length of the gradient vector and $g_{i,k}$ is the kth element of the gradient vector produced by the ith core. To alter this equation for handling an uneven distribution of the u matrix we need to make a change similar to the change made to the value calculation in order to handle this same problem. The new equation is

$$\nabla l_m(\theta|y)=\frac{\sum\limits_{i=1}^{r}j_i\ e^{v_i}\ g_{i,k}}{\sum\limits_{i=1}^{r}j_i\ e^{v_i}} \tag{10}$$

where j_i is the number of rows of the chunk of the u matrix being processed by the ith core, as mentioned previously. Then, for computational stability, to account for overflow and underflow, we can make a similar alteration to the one made above. Our final equation becomes

$$\nabla l_m(\theta|y)=\frac{\sum\limits_{i=1}^{r}j_i\ e^{v_i-a}\ g_{i,k}}{\sum\limits_{i=1}^{r}j_i\ e^{v_i-a}} \tag{11}$$

and a is defined as above. From this we know that

$$\frac{m}{r}\sum_{i=1}^{r} e^{v_i} = \sum_{i=1}^{r} j_i \; e^{v_i - a} \tag{12}$$

3.3.3 The Hessian

Finally, we approach the calculations for the hessian. First, we know that the equation to calculate the hessian is

$$\begin{aligned}\nabla^2 l_m(\theta|y) = & \frac{\sum_{k=1}^{m} \left[\nabla^2 \log f_\theta(u_k, y)\right] \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)}}{\sum_{k=1}^{m} \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)}} \\ & + \frac{\sum_{k=1}^{m} \left[\nabla \log f_\theta(u_k, y) - \nabla l_m(\theta|y)\right] \left[\nabla \log f_\theta(u_k, y) - \nabla l_m(\theta|y)\right]^T \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)}}{\sum_{k=1}^{m} \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)}}\end{aligned} \tag{13}$$

From equations 7 and 12, we know

$$\sum_{k=1}^{m} \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)} = \sum_{i=1}^{r} j_i \; e^{v_i - a} \tag{14}$$

Thus, we have

$$\begin{aligned}\nabla^2 l_m(\theta|y) = & \frac{\sum_{k=1}^{m} \left[\nabla^2 \log f_\theta(u_k, y)\right] \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)}}{\sum_{i=1}^{r} j_i \; e^{v_i - a}} \\ & + \frac{\sum_{k=1}^{m} \left[\nabla \log f_\theta(u_k, y) - \nabla l_m(\theta|y)\right] \left[\nabla \log f_\theta(u_k, y) - \nabla l_m(\theta|y)\right]^T \frac{f_\theta(u_k, y)}{\tilde{f}(u_k)}}{\sum_{i=1}^{r} j_i \; e^{v_i - a}}\end{aligned} \tag{15}$$

Notice that we can assure that the second part of equation 14 is accurate by providing $\nabla l_m(\theta|y)$, the gradient, to the function which calculates the hessian.

Last, we know $\frac{f_\theta(u_k, y)}{\tilde{f}(u_k)} = e^{b_k - a}$ where $b_k = \log f_\theta(u_k) + \log f_\theta(y|u_k) - \log \tilde{f}(u_k)$ so,

$$\nabla^2 l_m(\theta|y) = \frac{\sum_{k=1}^{m} \left[\nabla^2 \log f_\theta(u_k, y)\right] e^{b_k - a}}{\sum_{i=1}^{r} j_i \; e^{v_i - a}} + \frac{\sum_{k=1}^{m} \left[\nabla \log f_\theta(u_k, y) - \nabla l_m(\theta|y)\right] \left[\nabla \log f_\theta(u_k, y) - \nabla l_m(\theta|y)\right]^T e^{b_k - a}}{\sum_{i=1}^{r} j_i \; e^{v_i - a}} \quad (16)$$

Then, to calculate the final hessian for the complete u matrix we must sum the hessians produced by each core.

3.4 Closing the Cluster

The final step we take is to close the cluster. We can use `stopCluster` to accomplish this. The code then returns to completing computations using a single core.