Computational Statistics (2020) 35:1273–1289
https://doi.org/10.1007/s00180-019-00950-7

ORIGINAL PAPER

Parallel computing in linear mixed models

Fulya Gokalp Yavuz[1,2] · Barret Schloerke[1]

Received: 31 January 2018 / Accepted: 23 December 2019 / Published online: 7 January 2020

Abstract
In this study, we propose a parallel programming method for linear mixed models (LMM) generated from big data. A commonly used algorithm, expectation maximization (EM), is preferred for its use of maximum likelihood estimations, as the estimations are stable and simple. However, EM has a high computation cost. In our proposed method, we use a divide and recombine to split the data into smaller subsets, running the algorithm steps in parallel on multiple local cores and combining the results. The proposed method is used to fit LMM with dense and sparse parameters and for large number of observations. It is faster than the classical approach and generalizes for big data. Supplementary sources for the proposed method are available in the R package *lmmpar*.

Keywords Big data · Divide and recombine · EM · Linear mixed models · R package

1 Introduction

Linear mixed models (LMM) have been extensively used for repeated measurements, which emerge in longitudinal studies and clustered data. This type of model includes both fixed and random effects. The expectation maximization (EM) algorithm is used to carry out the parameter estimations of LMM and repeating the EM algorithm until convergence. The several, serially executed iterations of the algorithm cause computational burden and require efficient memory management, especially for big data sets. Recently, more sophisticated methods and speedup strategies have fostered an increase in the computational speed and decrease the required memory allocations.

✉ Fulya Gokalp Yavuz
fulyagokalp@gmail.com

1 Purdue University, 250 N. University St., West Lafayette, IN 47906, USA

2 Department of Statistics, Middle East Technical University, 140, Ankara, Turkey

Statistical methods are implemented in parallel within the compass of some of the current studies, albeit they are limited. In the study of Renaut (1998), multi-splitting methods are used in the least squares problems and the decomposed matrices are solved in parallel. In one of the recent studies, several parallel statistical methods are reviewed and the domain decomposition is used by defining the projector operators map vectors to split the data at some part of the study (Guo 2012). Maclaurin and Adams (2014) tries to speed up MCMC by defining a binary auxiliary variable with the conditional likelihood to decrease the evaluation time of likelihoods. Wolfe et al. (2008) introduces a fully distributed EM algorithm and uses it in three separate MapReduce topologies.

There are several studies, including parallel computing for basic statistics, but the analytics of complicated models are limited with big data in that memory and time constraints are major concerns. Parallelizing statistical algorithms efficiently has become crucial for faster performance given the gradual increase in data magnitude and the availability of multi-core environments. Development in the message-passing clusters brings more programming opportunities for statistical analysis.

'Divide and recombine' strategies with one of the R packages, named *plyr*, is introduced in Wickham (2011). In this package, a data set is broken up into smaller pieces. Each piece calculates a result independently and the results are combined back in main memory. The list of R packages including high-performance and parallel computing can be found in https://cran.r-project.org/web/views/HighPerformanceComputing.html.

Parallel programming used in this study adheres to the 'divide and recombine' strategy. In LMM content, Tran et al. (2016) used the same strategy with a different aspect. They developed a hybrid algorithm method in Bayesian framework for generalized LMM within a static setting. Broderick et al. (2013) developed a similar method to Tran et al. (2016), but with dynamic variables instead of static variables. Our study is similar to Tran et al. (2016) and Neiswanger et al. (2014), in terms of splitting the full data into smaller subsets, running the algorithm steps separately, and finally combining the results.

We prefer to use the R language for the implementation as R has fast vector calculations and numerous, open-source statistical model routines already available. Larger data sets, however, require more attention to carry out the data manipulations and the statistical analyses. Parallel computing is a great candidate to handle the increase in data size. The proposed method is implemented in R using *lmmpar* package which is designed by the authors for this type of models and it can be found in https://cran.r-project.org/package=lmmpar (Gokalp Yavuz and Schloerke 2017).

In the next section, we give a brief introduction about the LMM setting. One of the EM-type algorithms, ECME, is defined in the third section. We then introduce the parallel programming and our proposed parallel LMM algorithm in the fourth section. Simulation studies and conclusion are discussed at the fifth and sixth sections, respectively.

2 Linear mixed models

The LMM is defined for a continuous response variable as following

$$\begin{aligned} \mathbf{y}_i &= \mathbf{X}_i\boldsymbol{\beta} + \mathbf{Z}_i\mathbf{u}_i + \mathbf{e}_i, i = 1, 2 \ldots, n, \\ \mathbf{u}_i &\sim \mathrm{N}_q\left(\mathbf{0}, \boldsymbol{\psi}\right), \\ \mathbf{e}_i &\sim \mathrm{N}_{n_i}\left(\mathbf{0}, \mathbf{R}_i\right), \end{aligned} \tag{1}$$

where $\mathbf{y}_i$ denotes an n_i dimensional vector of continuous responses for the i^{th} subject, $\boldsymbol{\beta}$ denotes a p dimensional vector of unknown population parameters, $\mathbf{u}_i$ denotes a q dimensional vector of unknown individual effects, $\mathbf{X}_i$ is a known $(n_i \times p)$ design matrix, $\mathbf{Z}_i$ is a known $(n_i \times q)$ design matrix, $\mathbf{e}_i$ denotes an n_i dimensional vector of residual errors assumed to be independent of $\mathbf{u}_i$. $\boldsymbol{\psi}$ is a $(q \times q)$ covariance matrix of random effects; while $\mathbf{R}_i$ is a $(n_i \times n_i)$ covariance matrix of residual errors (Laird and Ware 1982). It is often assumed that $\mathbf{R}_i = \sigma^2\mathbf{I}_{n_i}$ for simplicity. σ^2 and $\boldsymbol{\psi}$ are positive definite.

The joint distribution of $\left(\mathbf{y}_i^T, \mathbf{u}_i^T\right)^T$ is

$$\begin{bmatrix} \mathbf{y}_i \\ \mathbf{u}_i \end{bmatrix} \sim \mathrm{N}_{n_i+q}\left(\begin{bmatrix} \mathbf{X}_i\boldsymbol{\beta} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{Z}_i\boldsymbol{\psi}\mathbf{Z}_i^T + \mathbf{R}_i & \mathbf{Z}_i\boldsymbol{\psi} \\ \boldsymbol{\psi}\mathbf{Z}_i^T & \boldsymbol{\psi} \end{bmatrix}\right). \tag{2}$$

The marginal distribution of the response variable is obtained as

$$\mathbf{y}_i \sim N_{n_i}\left(\mathbf{X}_i\boldsymbol{\beta}, \sigma^2\mathbf{W}_i^{-1}\right), \tag{3}$$

where $\mathbf{W}_i = (\mathbf{Z}_i\mathbf{D}\mathbf{Z}_i + \mathbf{R}_i)^{-1}$ and $\mathbf{D} = \sigma^{-2}\boldsymbol{\psi}$. The maximum-likelihood (ML) estimates of $\boldsymbol{\beta}, \sigma^2$ and $\mathbf{D}$ is found by maximizing the following log-likelihood function of (3).

$$\begin{aligned} \mathrm{L}_0\left(\boldsymbol{\beta}, \sigma^2, \mathbf{D}\right) \propto \left(\sigma^2\right)^{-N/2} \prod_{i=1}^{n} |\mathbf{W}_i|^{1/2} \\ exp\left\{-\frac{1}{2\sigma^2}\left(\mathbf{y}_i - \mathbf{X}_i\boldsymbol{\beta}\right)^T \mathbf{W}_i\left(\mathbf{y}_i - \mathbf{X}_i\boldsymbol{\beta}\right)\right\}, \end{aligned} \tag{4}$$

where $N = \sum_{i=1}^{n} n_i$.

Given $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$ and $\left(\boldsymbol{\beta}, \sigma^2, \mathbf{D}\right)$, the random effects, $\mathbf{u}_1, \ldots, \mathbf{u}_n$ are independently and normally distributed with the following moments

$$\mathrm{E}\left(\boldsymbol{u}_i | \boldsymbol{y}, \boldsymbol{\beta}, \sigma^2, \boldsymbol{D}\right) = \boldsymbol{U}_i\boldsymbol{Z}_i^T\boldsymbol{R}_i^{-1}\left(\boldsymbol{y}_i - \boldsymbol{X}_i\boldsymbol{\beta}\right), \tag{5}$$

$$\mathrm{V}\left(\boldsymbol{u}_i | \boldsymbol{y}, \boldsymbol{\beta}, \sigma^2, \boldsymbol{D}\right) = \sigma^2\boldsymbol{U}_i, \tag{6}$$

where

$$\boldsymbol{U}_i = \left(\boldsymbol{D}^{-1} + \boldsymbol{Z}_i^T\boldsymbol{R}_i^{-1}\boldsymbol{Z}_i\right)^{-1}. \tag{7}$$

Detailed proofs and empirical Bayes point and interval estimates for random effects are found in Schafer (1998).

3 ECME algorithm

Two methods are commonly used to maximize (4): Expectation Maximization (EM) and Newton-Raphson (NR). EM (Dempster et al. 1977) is easy and stable, and it is proven that the likelihood increases at each iteration of this algorithm. The computations for parameter estimations are tractable for the log-likelihood function with EM algorithm. Therefore, we prefer to use an EM-type algorithm to maximize the likelihood function of LMM.

The EM algorithm is composed of two steps. The first step is finding the conditional expectation of the complete data log-likelihood with respect to unknown data given the observed data and the current parameter estimations. The second step includes the maximization of this expectation. These two steps are repeated until the parameters have converged.

Using the general EM scheme, the LMM random effects are treated as missing data and their conditional expectations are evaluated with (5). During the maximization step, some of the unknown parameters are fixed while the remaining parameters are maximized first. Once optimal values are found, the originally fixed parameters are maximized. Thus, the algorithm is no longer EM, it is called Expectation/Conditional Maximization Either (ECME) (Liu and Rubin 1994).

The maximum likelihood parameters of LMM, are inferred by maximizing the log-likelihood function given in Eq. (4) with ECME algorithm. They are given below similar to the ones defined in Schafer (1998)

$$\boldsymbol{U}_i^{(k)} = \left(\boldsymbol{D}^{-1(k)} + \boldsymbol{Z}_i^T \boldsymbol{R}_i^{-1} \boldsymbol{Z}_i\right)^{-1}, \tag{8}$$

$$\boldsymbol{W}_i^{(k)} = \boldsymbol{R}_i^{-1} - \boldsymbol{R}_i^{-1} \boldsymbol{Z}_i \boldsymbol{U}_i^{(k)} \boldsymbol{Z}^T \boldsymbol{R}_i^{-1}, \tag{9}$$

$$\boldsymbol{\beta}^{(k)} = \left(\sum_{i=1}^{n} \boldsymbol{X}_i^T \boldsymbol{W}_i^{(k)} \boldsymbol{X}_i\right)^{-1} \left(\sum_{i=1}^{n} \boldsymbol{X}_i^T \boldsymbol{W}_i^{(k)} \boldsymbol{y}_i\right), \tag{10}$$

$$\sigma_i^{2(k+1)} = \frac{1}{\mathrm{N}} \sum_{i=1}^{n} \left(\boldsymbol{y}_i - \boldsymbol{X}_i \boldsymbol{\beta}^{(k)}\right) \boldsymbol{W}_i^{(k)} \left(\boldsymbol{y}_i - \boldsymbol{X}_i \boldsymbol{\beta}^{(k)}\right), \tag{11}$$

$$\boldsymbol{u}_i^{(k)} = \boldsymbol{U}_i^{(k)} \boldsymbol{Z}_i^T \boldsymbol{R}_i^{-1} \left(\boldsymbol{y}_i - \boldsymbol{X}_i \boldsymbol{\beta}^{(k)}\right), \tag{12}$$

$$\boldsymbol{D}^{(k+1)} = \frac{1}{n} \sum_{i=1}^{n} \left(\sigma^{-2(k)} \boldsymbol{u}_i^{(k)} \boldsymbol{u}_i^{(k)T} + \boldsymbol{U}_i^{(k)}\right). \tag{13}$$

The likelihood evaluated at cycle (k) is as follows

$$\mathrm{L}\left(\boldsymbol{\beta}^{(k)}, \sigma^{2(k)}, \boldsymbol{D}^{(k)}\right) = -\frac{N}{2}\log\sigma^{2(k)} - \frac{n}{2}\log\left|\boldsymbol{D}^{(k)}\right| + \frac{1}{2}\sum_{i=1}^{n}\log\left|\boldsymbol{U}_i^{(t)}\right| - \frac{N}{2}\left(\frac{\sigma^{2(k+1)}}{\sigma^{2(k)}}\right). \tag{14}$$

The ECME algorithm updates the unknown parameters $\boldsymbol{\beta}, \sigma^2, \boldsymbol{D}$ each step until (14) converges. Maximum likelihood estimates of $\boldsymbol{\beta}, \sigma^2$ and $\boldsymbol{D}$ will be consistent, if $n \to \infty$ and n_i remains bounded, with the difference of these estimates from the true parameters by terms of size $O_p(n^{-1/2})$ (Liu and Rubin 1994).

4 Parallel programming

With parallel programming, a computation-intensive process is divided into several parts or tasks. The complete set of tasks are able to be processed at a faster rate in parallel, as the workload is spread between multiple cores simultaneously. Detailed overview of the parallel methods in point of the statistical computing is found in Kontoghiorghes (2005).

A task in which each core does the same statistical job with different subsets of the data is called *domain decomposition*. Parallelization on the tasks of a statistical method is called *functional decomposition*. We use domain decomposition and implement the computations in parallel. Since the calculations are implemented over a big set of matrices, it is faster to split each matrix into smaller pieces and run the required calculations on multiple cores. The achievable speedup using domain decomposition alone is limited as the demand on the hardware usage is higher (Nagel and Rickert 2001).

Different worker processes can be currently managed by the *parallel* package (R Core Team 2017). Other common packages such as *plyr* and *foreach* (Ooi et al. 2019b) wrap around the *parallel* package with integration help from the package *doParallel* (Ooi et al. 2019a). *plyr* is used for its clean interface and *foreach* for determining which variables need to be exported for the algorithm to work properly.

One of the new packages designed for big matrices is *bigmemory* (Kane et al. 2013). Data is not duplicated to each process with *bigmemory*. Instead, each matrix is put into shared memory (with *big.matrix()*) that local cores may access. This memory management style requires less overall memory when using many local processes. Converting data into a *big.matrix()* optimizes the communication cost for the algorithm. As our algorithm only needs to read the *big.matrix()* data, each core may work independently on its own section of the problem without fear of concurrency issues.

More specifically, we generate a big column vector from $\boldsymbol{y}$, $\boldsymbol{X}$ and $\boldsymbol{Z}$ matrices by using *big.matrix* package in R similar to MapReduce programming model. In our algorithm, the subjects are allocated to shared memory, and the matrices are separated by each subject to shared memory. So, a heavy matrix is decomposed to many sub-matrices to perform the parallel processing. This part can be seen as the *Map* procedure. Following the decomposition, each sub-matrix is processed independently. This part can be seen as *Reduce* process.

It is worth to note that the processors do not communicate in our proposed algorithm, they send the related message to the main memory. The final calculations are implemented in this main memory with the received compound messages.

Before introducing our algorithm for LMM, we want to give some details about one of the concepts used to check the performance of the parallel computing at the next section.

4.1 Speedup for parallel computing

An increase in the number of processors does not guarantee a linear increase in speedup for parallel computing because of waiting time, cost of communication, and shared memory. Also, it may not be possible to parallelize the entire code, especially in the case of complex models. The serial part of the code causes some limitations. The speedup measurement is used to evaluate the performance advantage of the parallel computing.

Let S_p denotes the speedup;

$$S_p = \frac{T_s}{T_p}, \tag{15}$$

where T_s is the sequential algorithm execution time, and T_p is the parallel algorithm execution time with p processors.

In parallel computing, there are theoretical upper limits for the speedup. Amdahl's law is one of them. It is applied to predict the speedup with the usage of multiple processors. Amdahl's law states

$$S_p = \frac{1}{f_s + \frac{f_p}{C}}, \tag{16}$$

where f_s is the serial fraction of the code, f_p is the parallel fraction of the code, and C is the number of the processors.

The maximum number of cores used in this study is 16. Using Amdahl's law which shows the evaluation of speedup versus number of processors for different fractions of parallel part of the code, we do not expect more than 9x speedup for the parallel calculations. Keep in mind that 90% of our code is executed in parallel. (For the Amdahl's law, please refer: https://en.wikipedia.org/wiki/Amdahl' s_law)

4.2 Linear mixed models with parallel programming

The ECME algorithm introduced in the previous section is implemented using the R language with a 'divide and combine' approach. Steps 8, 9, and 12 are implemented for each subject, and steps 10, 11, and 13 are mainly average value combinations of these values with observed data. We aim to divide these calculations and process the corresponding segments of the algorithm in parallel.

We perform calculations of 8, 9, and 12 on each core, $c = 1, 2, \ldots, C$, where C is the maximum number of cores and $C < n$. In other words, the data is divided into

C parts and the E-step is executed on different cores as each subject is independent given a current estimate. (Repeats are dependent, since they are gathered from the same subject).

Let η and $\boldsymbol{\theta} = \left(\boldsymbol{\beta}, \sigma^2, \boldsymbol{D}\right)$ denote the sufficient statistics and parameters, respectively. The data $\boldsymbol{A}$ is divided into C splits and the sufficient statistics for the given parameters are:

$$\eta^{(i)} = \mathrm{E}_\theta\left(\eta|\boldsymbol{A}_i\right). \tag{17}$$

For the first core, the first sufficient statistic is calculated as

$$\eta_1^{(i)} = \boldsymbol{X}_i^T \boldsymbol{W}_i^{(current)} \boldsymbol{X}_i, \tag{18}$$

where $\boldsymbol{W}_i$ is defined in Eq. 3. The parallel sufficient statistic for the first combination is

$$\eta_1^{(c)} = \sum_{i=1}^{N/C} \boldsymbol{X}_i^T \boldsymbol{W}_i^{(current)} \boldsymbol{X}_i. \tag{19}$$

Note that for the jth process, i is defined from $[(j-1)\frac{\mathrm{N}}{C}+1]$ to $[j\frac{\mathrm{N}}{C}]$. For the ease of notation, we assume that the number of processes is an exact divisor of the total number of observations. If not, the system optimally sets it.

The final combination for the first sufficient statistics is

$$\eta_1 = \sum_{j=1}^{C} \eta_1^{(j)}. \tag{20}$$

Depending on the notation of (20), $\hat{\boldsymbol{\beta}}$ will be

$$\hat{\boldsymbol{\beta}} = \left(\eta_1\right)^{-1} \eta_2. \tag{21}$$

where

$$\eta_2 = \sum_{j=1}^{C} \eta_2^{(j)}, \quad \text{and} \quad \eta_2^{(c)} = \sum_{i=1}^{N/C} \boldsymbol{X}_i^T \boldsymbol{W}_i^{(current)} \boldsymbol{y}_i.$$

.

More explicitly, the parallel maximum likelihood estimator (PMLE) in a linear mixed model is

$$\tilde{\boldsymbol{\beta}} = \left(\sum_{j=1}^{C} \sum_{i=[(j-1)\frac{\mathrm{N}}{C}+1]}^{[j\frac{\mathrm{N}}{C}]} \boldsymbol{X}_{ji}^T \hat{\boldsymbol{W}}_{ji} \boldsymbol{X}_{ji}\right)^{-1} \left(\sum_{j=1}^{C} \sum_{i=[(j-1)\frac{\mathrm{N}}{C}+1]}^{[j\frac{\mathrm{N}}{C}]} \boldsymbol{X}_{ji}^T \hat{\boldsymbol{W}}_{ji} \boldsymbol{y}_{ji}\right). \tag{22}$$

Similarly, variance covariance matrices for error and random terms are defined as below:

$$\hat{\sigma}^2 = \frac{1}{N} \sum_{j=1}^{C} \sum_{i=[(j-1)\frac{\mathrm{N}}{C}+1]}^{[j\frac{\mathrm{N}}{C}]} \left(\boldsymbol{y}_{ji} - \boldsymbol{X}_{ji}\hat{\boldsymbol{\beta}}\right) \hat{\boldsymbol{W}}_{ji} \left(\boldsymbol{y}_{ji} - \boldsymbol{X}_{ji}\hat{\boldsymbol{\beta}}\right), \tag{23}$$

$$\hat{\boldsymbol{D}} = \frac{1}{n}\sum_{j=1}^{C}\sum_{i=[(j-1)\frac{N}{C}+1]}^{[j\frac{N}{C}]}\left(\hat{\sigma}^2\boldsymbol{u}_i\boldsymbol{u}_i^T + \boldsymbol{U}_i\right). \tag{24}$$

Note that (22), (23), and (24) include the estimations of $\boldsymbol{U}_i$, $\boldsymbol{W}_i$, and $\boldsymbol{u}_i$ which are defined in (8), (9), and (12), respectively. $\boldsymbol{U}_i$ is used to find the variance-covariance matrix of random effects. $\boldsymbol{W}_i$ is the variance-covariance matrix of the response variable and $\boldsymbol{u}_i$ is the random effect predictions of the i^{th} subject.

Assume that $\tilde{\boldsymbol{\beta}}$ in 22 can be written as

$$\tilde{\boldsymbol{\beta}} = \frac{1}{C}\sum_{j=1}^{C}\hat{\boldsymbol{\beta}}_j \tag{25}$$

where C is the number of cores and $\hat{\boldsymbol{\beta}}_j$ is defined similar to the equation (10) for each core. Then, the similar statistical properties of the parameter can be found in the study of Guo et al. (2015).

As defined previously, the ECME algorithm is used for our estimations and the parameter estimations are found using (22), (23), and (24). The general shape of the ECME algorithm for all of the parameters is implemented in pseudo code below:

Algorithm 1 Parallel Linear Mixed Models

```
Require: $\boldsymbol{D}$ and $\sigma^2$ are positive definite
Ensure: $\boldsymbol{y}$, $\boldsymbol{X}$, and $\boldsymbol{Z}$ inherit big.matrix        ▷ $\boldsymbol{y}_{big}$, $\boldsymbol{X}_{big}$, $\boldsymbol{Z}_{big}$
1: function lmmpar
2:     $\boldsymbol{\theta}_0 \leftarrow$ initials
3:     Register cores for parallel processing
4:     while Convergence has not been met do
5:         if $cores = 1$ then                              ▷ Calling function
6:             Call $core\_fn()$                            ▷ Non-parallel
7:         else
8:             Call $core\_fn()$ in parallel on registered cores   ▷ Parallel
9:         end if
10:        Combine sufficient statistics
11:        Check that $\boldsymbol{D}$ and $\sigma^2$ are positive definite
12:    end while
13:    return $\boldsymbol{\beta}$, $\boldsymbol{D}$, and $\sigma^2$
14: end function
15:
16: function $core\_fn$                                   ▷ Processing function
17:    Retrieve the indicies this particular process should use
18:    for $i$ in $indicies$ do
19:        Subset $\boldsymbol{y}_{big}$, $\boldsymbol{X}_{big}$, and $\boldsymbol{Z}_{big}$ according to $subject[i]$
20:        Calculate the sufficient statistics and required fields for parameter estimations
21:    end for
22:    Locally combine sufficient statistics
23:    return Sufficient statistics
24: end function
```

5 Simulation study

The aim of this section is to validate the efficiency of the proposed parallel computing in LMM. The LMM is defined as:

$$\begin{aligned} \mathbf{y}_i &= \mathbf{X}_i\boldsymbol{\beta} + \mathbf{Z}_i\mathbf{u}_i + \mathbf{e}_i, i = 1, 2 \ldots n, \\ \mathbf{u}_i &\sim \mathrm{N}_q(\mathbf{0}, \boldsymbol{\psi}), \\ \mathbf{e}_i &\sim \mathrm{N}_{n_i}(\mathbf{0}, \mathbf{R}_i). \end{aligned} \tag{26}$$

The definitions of all terms in this model are the same as the model in (1). This is a subject specific model including a random effect $\boldsymbol{u}_i$, which is predicted (estimated) for each subject. In the model, i denotes a subject and i^{th} subject has n_i repeats. Each subject is independent of each other while a dependency exists between repeats. For the ease of the computation, all vectors and matrices are stacked vertically. As an example, the response variable is defined as $\boldsymbol{y} = (\boldsymbol{y}_1^T, \boldsymbol{y}_2^T, \ldots, \boldsymbol{y}_n^T)^T$, where $\boldsymbol{y}_i = (y_{i1}, y_{i2}, \ldots, y_{in_i})$.

Alternatively $\boldsymbol{X}$ and $\boldsymbol{Z}$ matrices can be defined as arrays for fixed and random effects, respectively. With this design, they store n rectangular matrices each with n_i rows and p or q columns for $\boldsymbol{X}$ and $\boldsymbol{Z}$, respectively. We prefer to use the matrix format, as the 'bigmemory' package is appropriate for matrices and vectors.

Simulation studies were conducted with the R package *lmmpar*, which was designed for this study. When LMM was run with 'lmm' package, the computations become unfeasible with the increase in the number of parameters.

Dense and sparse $\boldsymbol{\beta}$ parameters are used in the simulation model to see the performances for both situations. The dense vector of $\boldsymbol{\beta}$ is generated from the normal distribution with mean 10 and variance 1. The sparse vector of $\boldsymbol{\beta}$ is generated from the binomial distribution with 0.1 probability. Both dense and sparse parameters are common in application, so the performance of each situation should be tested. $\mathbf{R}_i = \sigma^2\mathbf{I}_{n_i}$ and $\sigma^2 = 1$ for simplicity. The variance-covariance matrix of random effects varies for different scenarios. For example, for 2×2 matrix, it is taken as

$$\boldsymbol{D} = \begin{bmatrix} 16 & 0 \\ 0 & 0.025 \end{bmatrix}.$$

Three main components affect the performance of a parallel computing: the size of vectors and matrices, the data layout, and the number of cores. So, the number of cores, observations, and parameters are altered to see the change in the speed of calculations in the model (26).

We generate the data with different conditions by assigning alternative values for n and p, such as $n = 10^5, 10^6$, and 2×10^6 and $p = 5$ and 51 for each scenario. Our algorithm is expected to run faster than existing algorithms as the data size increases. Data size was enlarged by increasing the number of observations and parameters.

Figure 1 depicts the number of cores versus elapsed time for dense and sparse $\boldsymbol{\beta}$ and for different values of n and p. Columns of the panels define the number of

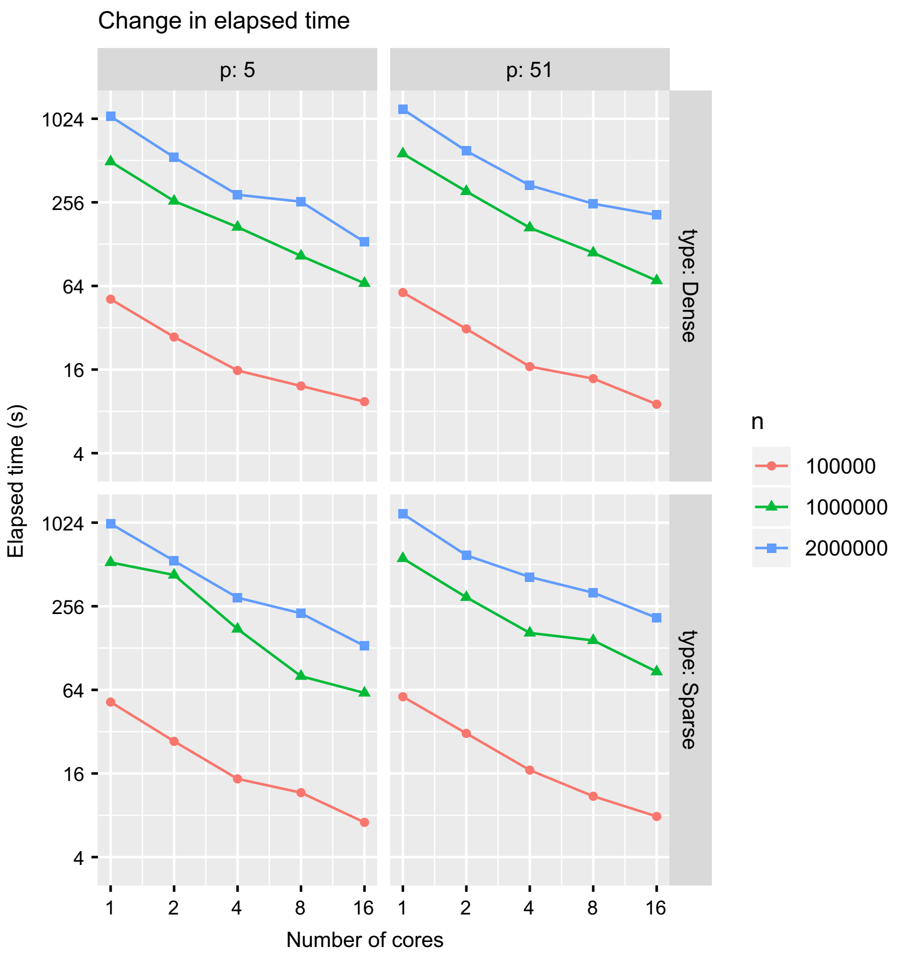

Fig. 1 Number of cores versus elapsed time (*log2 formatting on y-scales*)

variables while rows define the type of $\boldsymbol{\beta}$. It is named as *dense*, if $\boldsymbol{\beta}$ is generated from the normal distribution and it is named as *sparse*, if $\boldsymbol{\beta}$ is generated from the binomial distribution.

The decrease in the elapsed time is obvious with the increase in the number of cores used for parallel computing in Fig. 1. The speedup graph, which shows the performance of the executed code for different scenarios, is reported in Fig. 2. The purple line is the reference, which shows the ideal situation. The sub-linearity is the expected area for a parallel computing. According to Amdahl's Law, it is not expected to have more than our achieved $9\times$ speedup, even for the best case scenario.

Table 1 reports the *system.time* results and parameters used, most importantly, the user time, user child time, elapsed time, cores used, observations (n), and number of parameters (p) for the dense $\boldsymbol{\beta}$. The elapsed time is used to find the speed up and to draw Fig. 1.

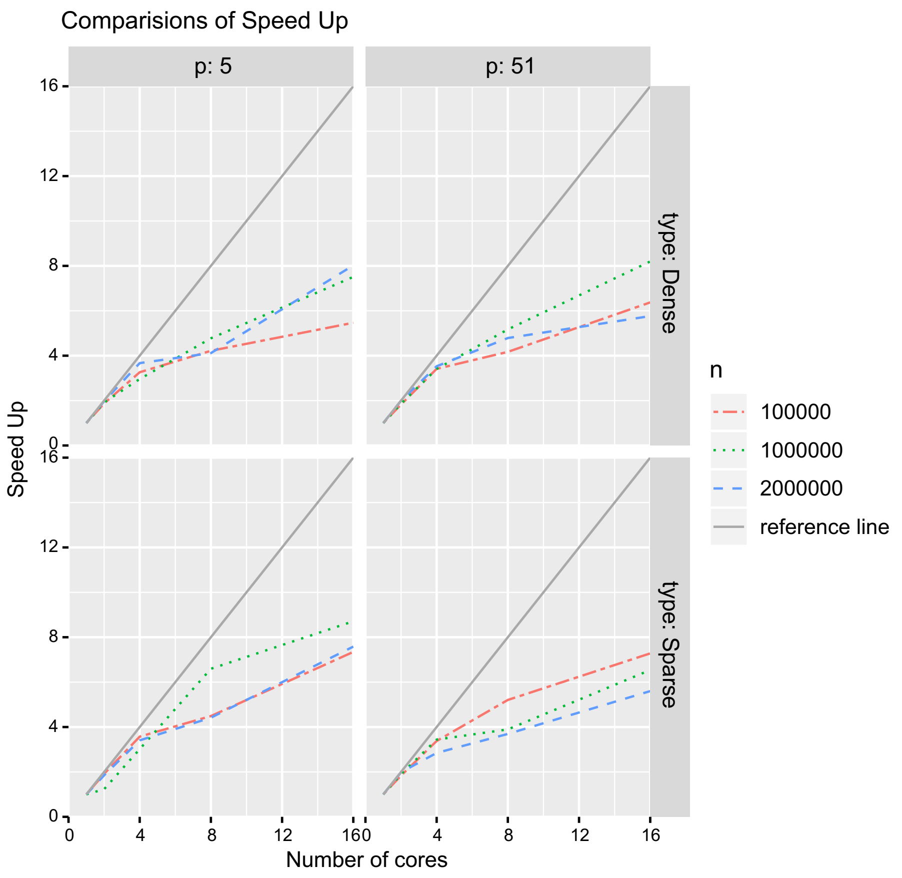

Fig. 2 Speedup versus number of cores

In the simulation set with one million observations, it takes just under 67 s with 16 cores, while it takes over 500 s without parallel computing to complete the same job.

The initial values of $\boldsymbol{\beta}$ and the parameter estimations are reported in Table 2 for the dense parameter scenario. These values are used to see how well the model fits the data. The estimated parameter values in Table 2 are within a reasonable tolerance of the true values.

Parameter estimations in model (26) with small dimensions are reported in Tables 2 and 5 for dense and sparse $\boldsymbol{\beta}$, respectively. On the other hand, we preferred to report the difference between the initials and the estimations for higher dimensional parameters for the ease of readability. Figure 3 depicts these differences for dense $\boldsymbol{\beta}$ in three different number of observation scenarios. The spread in differences shrinks with the increase in the number of observations.

Table 3 includes the variance covariance estimations for random effects and variance estimation for the error terms for dense $\boldsymbol{\beta}$.

Table 1 Simulation results for dense beta

Elapsed	User.self	Sys.self	User.child	Sys.child	Cores	n	p
9.400	1.608	0.232	63.652	4.920	16	10^5	5
12.182	0.948	0.100	56.276	2.600	8	10^5	5
15.750	0.828	0.052	54.156	1.968	4	10^5	5
27.441	1.084	0.044	44.480	0.892	2	10^5	5
51.390	50.976	0.332	0	0	1	10^5	5
66.903	13.316	1.364	531.852	43.864	16	10^6	5
105.233	7.016	0.584	543.056	33.040	8	10^6	5
169.757	6.888	0.472	498.624	19.028	4	10^6	5
261.509	6.852	0.368	432.320	7.832	2	10^6	5
502.392	499.536	2.604	0	0	1	10^6	5
132.760	16.048	3.028	1142.880	133.012	16	2×10^6	5
258.797	15.272	2.008	1092.420	69.340	8	2×10^6	5
290.270	15.136	1.720	905.352	35.048	4	2×10^6	5
540.048	15.056	1.820	1030.816	28.324	2	2×10^6	5
1065.113	1029.456	33.968	0	0	1	2×10^6	5
9.004	3.180	0.760	63.712	7	16	10^5	51
13.765	1.264	0.264	62.736	5.020	8	10^5	51
16.811	1.160	0.268	53.464	2.772	4	10^5	51
31.422	1.156	0.196	49.272	2.344	2	10^5	51
57.395	56.908	0.436	0	0	1	10^5	51
69.905	10.912	2.820	635.696	104.528	16	10^6	51
110.850	10.496	2.892	574.784	42.180	8	10^6	51
168.051	10.540	2.300	548.424	32.992	4	10^6	51
306.854	10.404	2.408	489.440	20.564	2	10^6	51
573.096	565.756	6.908	0	0	1	10^6	51
207.635	22.460	7.308	1413.276	443.424	16	2×10^6	51
250.023	22.052	6.552	1220.988	178.776	8	2×10^6	51
339.063	22.132	5.324	1077.832	102.648	4	2×10^6	51
600.221	22.068	5.056	959.636	45.776	2	2×10^6	51
1196.785	1169.588	25.456	0	0	1	2×10^6	51

The same scenarios are repeated for sparse $\boldsymbol{\beta}$ to see how the algorithm performs with sparse matrices which are highly common in big data. The simulation results are located in Table 4. Parameter estimations are reported in Table 5 for sparse $\boldsymbol{\beta}$. Similar to the dense $\boldsymbol{\beta}$ scenario, the differences between the initial and estimated parameters are plotted for three different number of observations. Differences are found in Fig. 4. The variance estimations are found in Table 6 for the detailed investigation of the readers.

Table 2 Parameter estimations for Dense $\boldsymbol{\beta}$, $p = 5$

n	$\boldsymbol{\beta}_0$	$\hat{\boldsymbol{\beta}}$	$\hat{\boldsymbol{D}}$	$\hat{\sigma}^2$
10^5	$\begin{bmatrix} 1.000 \\ 9.999 \\ 9.789 \\ 8.001 \\ 9.192 \end{bmatrix}$	$\begin{bmatrix} 0.987 \\ 9.999 \\ 9.787 \\ 8.002 \\ 9.191 \end{bmatrix}$	$\begin{bmatrix} 15.9503 & 0.0004 \\ 0.0004 & 0.0249 \end{bmatrix}$	1.000
10^6	$\begin{bmatrix} 1.000 \\ 10.941 \\ 10.487 \\ 8.860 \\ 9.610 \end{bmatrix}$	$\begin{bmatrix} 0.998 \\ 10.943 \\ 10.487 \\ 8.860 \\ 9.610 \end{bmatrix}$	$\begin{bmatrix} 16.0034 & -0.0003 \\ -0.0003 & 0.0250 \end{bmatrix}$	0.999
2×10^6	$\begin{bmatrix} 1.000 \\ 11.519 \\ 9.655 \\ 9.744 \\ 9.520 \end{bmatrix}$	$\begin{bmatrix} 0.994 \\ 11.518 \\ 9.655 \\ 9.743 \\ 9.520 \end{bmatrix}$	$\begin{bmatrix} 16.0085 & -0.0004 \\ -0.0004 & 0.0250 \end{bmatrix}$	0.999

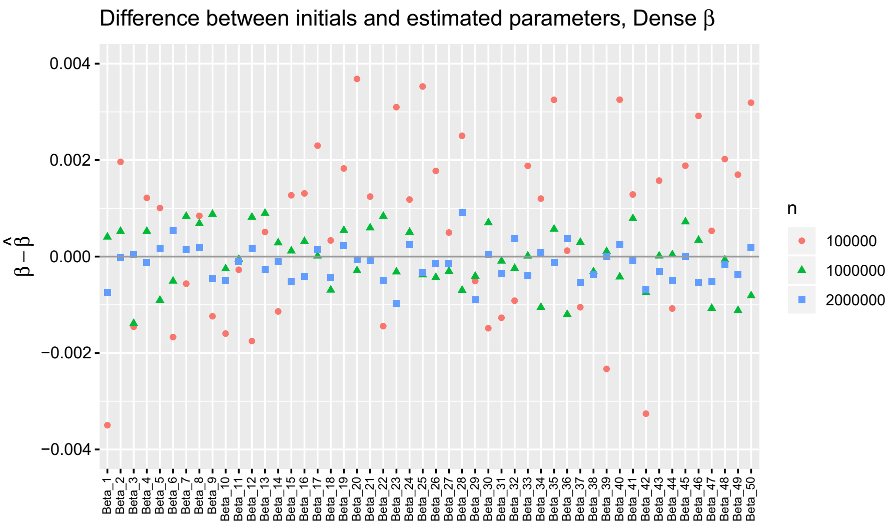

Fig. 3 Difference between initials and estimations, Dense Beta

Table 3 Variance-covariance estimations for dense and big $\boldsymbol{\beta}$, $p = 51$

n	$\hat{\boldsymbol{D}}$	$\hat{\sigma}^2$
10^5	$\begin{bmatrix} 16.0594 & -0.0008 \\ -0.0008 & 0.0249 \end{bmatrix}$	0.998
10^6	$\begin{bmatrix} 15.9844 & -0.0003 \\ -0.0003 & 0.0250 \end{bmatrix}$	0.999
2×10^6	$\begin{bmatrix} 16.0092 & 0.0001 \\ 0.0001 & 0.0250 \end{bmatrix}$	0.999

Table 4 Simulation results for sparse beta

Elapsed	User.self	Sys.self	User.child	Sys.child	Cores	n	p
7.116	1.400	0.232	60.208	5.588	16	10^5	5
11.642	1.012	0.096	56.728	3.316	8	10^5	5
14.629	0.884	0.064	48.444	1.604	4	10^5	5
27.267	0.880	0.048	44.528	1.152	2	10^5	5
52.273	52.020	0.176	0	0	1	10^5	5
60.786	7.788	1.616	555.756	68.652	16	10^6	5
80.274	7.228	1.232	482.168	32.972	8	10^6	5
175.483	7.028	0.820	507.168	21.636	4	10^6	5
429.789	6.996	0.864	475.284	12.640	2	10^6	5
529.440	515.116	13.484	0	0	1	10^6	5
132.608	15.600	1.908	1133.096	122.884	16	2×10^6	5
227.669	15.096	1.628	1081.032	70.428	8	2×10^6	5
295.135	14.972	1.560	996.548	42.168	4	2×10^6	5
543.268	14.400	1.184	878.696	24.428	2	2×10^6	5
1005.547	995.348	9.496	0	0	1	2×10^6	5
7.848	1.780	0.460	63.612	6.892	16	10^5	51
10.972	1.276	0.376	61.080	5.004	8	10^5	51
16.913	1.172	0.228	53.972	2.796	4	10^5	51
31.112	1.176	0.232	48.572	2.160	2	10^5	51
57.135	56.692	0.380	0	0	1	10^5	51
86.478	10.888	2.984	620.688	83.208	16	10^6	51
145.229	10.504	2.956	645.572	45.800	8	10^6	51
164.435	10.648	2.440	544.684	35.240	4	10^6	51
296.992	10.412	2.684	480.308	20.604	2	10^6	51
565.672	558.236	6.920	0	0	1	10^6	51
211.255	22.680	7.652	1422.776	376.984	16	2×10^6	51
320.383	21.936	6.624	1285.288	189.580	8	2×10^6	51
415.702	21.736	7.240	1142.424	84.436	4	2×10^6	51
595.867	22.072	6.352	1015.004	53.524	2	2×10^6	51
1183.315	1157.012	24.688	0	0	1	2×10^6	51

Table 5 Parameter estimations for Sparse $\boldsymbol{\beta}$, $p = 5$

n	$\boldsymbol{\beta}_0$	$\hat{\boldsymbol{\beta}}$	$\hat{\boldsymbol{D}}$	$\hat{\sigma}^2$
10^5	$\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0.981\\-0.001\\-0.004\\-0.001\\-0.000\end{bmatrix}$	$\begin{bmatrix}15.9608 & -0.0002\\-0.0002 & 0.0249\end{bmatrix}$	1.002
10^6	$\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}1.002\\0\\-0.006\\0\\0\end{bmatrix}$	$\begin{bmatrix}16.0006 & 0\\0 & 0.0250\end{bmatrix}$	1.000
2×10^6	$\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}1.003\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}16.015 & 0.0001\\0.0001 & 0.0249\end{bmatrix}$	0.999

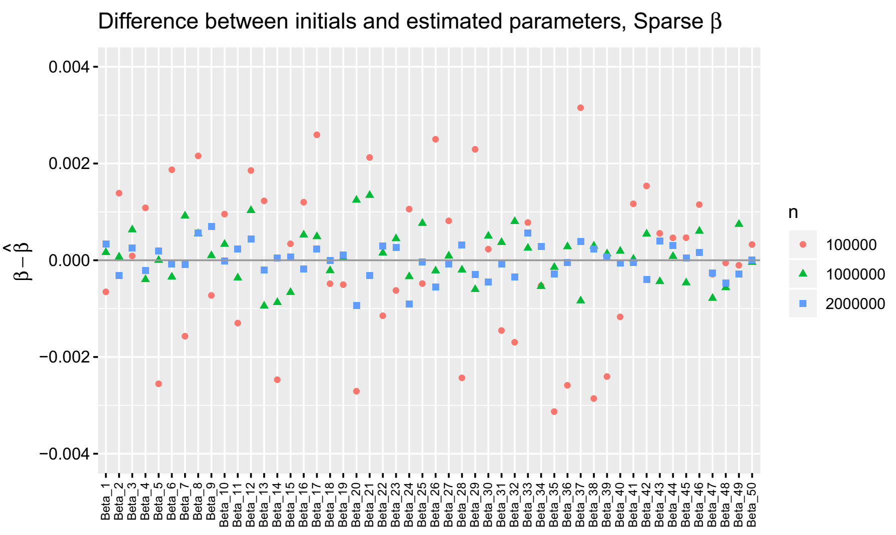

Fig. 4 Difference between initials and estimations, Sparse Beta

Table 6 Variance-covariance estimations for sparse and big $\boldsymbol{\beta}$, $p = 51$

n	$\hat{\boldsymbol{D}}$	$\hat{\sigma}^2$
10^5	$\begin{bmatrix}15.9822 & -0.0007\\-0.0007 & 0.0249\end{bmatrix}$	1.001
10^6	$\begin{bmatrix}16.0011 & 0.0003\\0.0003 & 0.0249\end{bmatrix}$	1.001
2×10^6	$\begin{bmatrix}15.9887 & 0\\0 & 0.0249\end{bmatrix}$	1.000

6 Conclusion and discussion

This study is a contribution to the application of parallel programming for complex statistical analyses using big data. We use the EM algorithm for the maximum likelihood estimations of a LMM with a continuous response variable, as it is easy and stable. However, it requires a large amount of memory for big data and is a highly time consuming procedure.

Simulation results show that the elapsed time of the *lmmpar* function is faster than the classical approach with a single core for all scenarios. This gives us a modeling flexibility for LMM with big data. The modeling approach can easily be extended for other types of statistical models.

It is crucial to find the better strategies for complex statistical models, since we see throughout our studies that parallel computing does not always guarantee to speedup in processing time or saving in memory usage. Finding an adequate strategy is the main aim in parallel computing programming for complex models. A limitation of this study is that the algorithm requires initial values and they are gathered from the *basic* model. We use initial values of the parameters for the simulated data.

This method is defined for normally distributed random effects and error terms, but it can easily be extended for more robust models such as t-LMM and Laplace-LMM. We also plan to extend the study for different settings of LMM (Yavuz and Arslan 2018; Pinheiro et al. 2001) to see the robustness of the parameters. Also, we are planning to extend the study to utilize the Apache Hadoop environment. Additionally, simulation results for the sparse matrices of $\boldsymbol{\beta}$ are negligible for the zero values of the parameters. This naturally extends to include a variable selection method for sparse parameters. We plan to use a LASSO-type variable selection method for the follow-up study.

References

Broderick T, Boyd N, Wibisono A, Wilson AC, Jordan MI (2013) Streaming variational Bayes. In proceedings of the 26th international conference on neural information processing systems—volume 2, NIPS'13. Curran Associates Inc, New York, pp 1727–1735

Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J Roy Stat Soc B 39(1):1–38

Gokalp Yavuz F, Schloerke B (2017) Parallel Linear Mixed Model https://CRAN.R-project.org/package=lmmpar, R package version 0.1.0

Guo G (2012) Parallel statistical computing for statistical inference. J Stat Theory Pract 6(3):536–565

Guo G, You W, Qian G, Shao W (2015) Parallel maximum likelihood estimator for multiple linear regression models. J Comput Appl Math 273:251–263

Kane MJ, Emerson J, Weston S (2013) Scalable strategies for computing with massive data. J Stat Softw 55(14):1–19

Kontoghiorghes EJ (2005) Handbook of parallel computing and statistics (statistics, textbooks and monographs). Chapman & Hall/CRC, Boca Raton

Laird NM, Ware JH (1982) Random-effects models for longitudinal data. Biometrics 38(4):963–74

Liu C, Rubin DB (1994) The ECME algorithm: A simple extension of EM and ECM with faster monotone convergence. Biometrika 81(4):633

Maclaurin D, Adams RP (2014) Firefly monte carlo: exact MCMC with subsets of data. In Proceedings of the thirtieth conference on uncertainty in artificial intelligence, UAI'14. AUAI Press, Arlington, pp 543–552

Nagel K, Rickert M (2001) Parallel implementation of the transims micro-simulation. Parallel Comput 27:1611–1639

Neiswanger W, Wang C, Xing EP (2014) Asymptotically exact, embarrassingly parallel MCMC. In Proceedings of the thirtieth conference on uncertainty in artificial intelligence, UAI'14. AUAI Press, Arlington, pp 623–632

Ooi H, Microsoft Corporation, Weston S, Tenenbaum D (2019a) doParallel: foreach parallel adaptor for the 'parallel' package. R package version 1.0.15. https://cran.r-project.org/web/packages/doParallel/index.html

Ooi H, Microsoft Corporation, Weston S (2019b) Foreach: provides foreach looping construct. R package version 1.4.7. https://cran.r-project.org/web/packages/foreach/index.html

Pinheiro JC, Liu C, Wu YN (2001) Efficient algorithms for robust estimation in linear mixed-effects models using the multivariate t distribution. J Comput Graph Stat 10(2):249–276

R Core Team (2017) R: a language and environment for statistical computing. R Foundation for Statistical Computing, Vienna

Renaut RA (1998) A parallel multisplitting solution of the least squares problem. Numer Linear Algeb Appl 5(1):11–31

Schafer JL (1998) Some improved procedures for linear mixed models. Technical Report, Department of Statistics, The Pennsylvania State University

Tran M-N, Nott DJ, Kuk AYC, Kohn R (2016) Parallel variational Bayes for large datasets with an application to generalized linear mixed models. J Comput Graph Stat 25(2):626–646

Wickham H (2011) The split-apply-combine strategy for data analysis. J Stat Softw 40(1):1–29

Wolfe J, Haghighi A, Klein D (2008) Fully distributed EM for very large datasets. In: Proceedings of the 25th international conference on machine learning, ICML '08. ACM, New York, pp 1184–1191

Yavuz FG, Arslan O (2018) Linear mixed model with Laplace distribution (LLMM). Stat Pap 59(1):271–289