Design Document for Parallelization R Package glmm

Sydney Benson

April 25, 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 gradient and hessian for use with MCLA.

1 The Packages

1.1 `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.

1.2 `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`.

1.3 `itertools`

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

2 The Process

2.1 Preparing the Cluster

2.1.1 `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.

2.1.2 `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.

2.1.3 `registerDoParallel`

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

2.1.4 `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`.

2.1.5 `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.

2.2 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 m̆atrix row-wise into parts, the number of parts being determined by the number of cores in our cluster.

2.3 Calculating the Hessian and Gradient

To calculate the hessian and gradient, we have to send each of our chunks of the m̆atrix through the `.C` function. We can do this using the `foreach` function and the `%dopar%` operator. We then have the hessian and gradient values from each chunk of the m̆atrix returned to us in a list. We can add the elements of this list to find our hessian and gradient.

2.4 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.