Multiple Imputation in Parallel with `parlMICE`

Gerko Vink & Rianne Schouten

Thursday, November 17th, 2016

For big datasets or high number of imputations, performing multiple imputation with function `mice` from package `mice`(Stef2011) might take long. As a solution, wrapper function `parlMICE` was created to enable the imputation procedure to be run in parallel. This is done by dividing the imputations over multiple cores (or CPUs), thereby speeding up the process.

Effect

In Figure 1, the processing time is shown for both `mice` and `parlMICE`. Here, dataset `nhanes` is imputed with `m` running from 18 to 3000. From the figure, it becomes apparent that for a small number of imputations, the conventional `mice` function works faster. From 330 imputations and more, wrapper function `parlMICE` is the most optimal function.

Put figure here.

In the last section of this vignette, the trade-off issue is discussed more deeply. For now we will dig into the arguments of function `parlMICE`.

Default settings

Imputation of a dataset (say, `nhanes`) can be performed with `parlMICE` in the following way.

```r
imp <- parlMICE(nhanes)
summary(imp)
```

```
## Multiply imputed data set
## Call:
## [[1]]
## mice(data = data, m = n.imp.core * n.core, printFlag = FALSE)
##
## [[2]]
## ibind(x = imp, y = imps[[i]])
##
## [[3]]
## ibind(x = imp, y = imps[[i]])
##
## Number of multiple imputations:  18
## Missing cells per column:
## age bmi hyp chl
##   0   9   8  10
## Imputation methods:
##   age   bmi   hyp   chl
##    "" "pmm" "pmm" "pmm"
## VisitSequence:
## bmi hyp chl
##   2   3   4
## PredictorMatrix:
```

```
##     age bmi hyp chl
## age   0   0   0   0
## bmi   1   0   1   1
## hyp   1   1   0   1
## chl   1   1   1   0
## Random generator seed value:  NA
```

The function returns a `mids` object as we know it from `mice`. Due to the parallel processing, the `call` of the object has slightly changed.

Using `mice` arguments

Function parlMICE is able to deal with the conventional `mice` arguments. In order to change the imputation method from its default (predictive mean matching) to, for example, Bayesian linear regression, the `method` argument is set to `"norm"`.

```r
imp <- parlMICE(nhanes, method = "norm")
```

The number of imputations can be given to `mice` with argument `m`. In `parlMICE` this is possible as well, but we advise to rather use the arguments `n.core` and `n.imp.core` for this. The next section discusses these arguments.

Arguments `n.core`, `n.imp.core` and `m`

With `n.core`, the number of cores (or CPUs) is defined. `n.imp.core` defines the number of imputations per core. Subsequently, the total number of imputations is defined by `n.core` * `n.imp.core`. As a default, `n.core` is specified as the number of available, logical cores minus 1. The default number of imputations per core has been set to 2.

Therefore, running `parlMICE` with its defaults on a computer with

```r
detectCores()
```

```
## [1] 4
```

available, logical cores, will result to `m = (4 - 1) * 2 = 6` imputations. Thus,

```r
imp <- parlMICE(nhanes, n.core = 2, n.imp.core = 10)
```

has the same number of imputations as

```r
imp <- mice(nhanes, m = 20)
```

Note that there is a trade-off between the number of cores that are used and the number of imputations per core. Logically, it takes working time to divide the imputations over the cores. Besides, the parallel imputed datasets need to be merged afterwards. Therefore, using too many cores might not be desirable.

However, for a large number of imputations one would like to use multiple cores, which is exactly the purpose of this function. Figure ref showed the trade-off for a dataset of 1000 cases and 4 variables, with 80 percent MCAR missingness. It seems reasonable that for larger datasets with more specific missingness or with a higher missingness proportion, running `mice` in parallel with `parlMICE` becomes desirable.

For small datasets, as `nhanes` is, we advise to use `parlMICE` for about 300 imputations and more. Figure 1 has shown this.
For big datasets…

Argument `seed`

In simulation studies, it is often desired to set a seed to make the results reproducible. In contrast to `mice`, the seed value for `parlMICE` cannot be defined outside the function. This has to do with the different cores running at the same time (for more information we gladly refer to the `parallel` manual).

A seed value can be used in `parlMICE` with argument `seed`:

```r
imp <- parlMICE(nhanes, seed = 123)
```

Mac and Windows

Function `parlMICE` calls for function `parLapply` from the `parallel` package. Although other options are available, we have chosen for `parLapply` because it allows the use of multiple cores on all computers, including a Windows computer. For the cluster, we rely on the default option in the function `makeCluster`, which is `"PSOCK"`.

On systems other than Windows, cluster type `"FORK"` might be faster (see the document written by Max Gordon). Therefore, we advise to change the type to `"FORK"` if your computer does not run on Windows.

```r
imp <- parlMICE(nhanes, type = "FORK")
```

References

Parallel manual

Gordon, M. (2015). How-to go parallel in R – basics + tips. Available with this link

Stef2011