Wrapper function `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` (Van Buuren and Groothuis-Oudshoorn 2011) might take a long time. 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.

First, this vignette will show the time gain with `parlMICE` for a small and (relatively) large dataset. Second, `parlMICE`'s arguments will be disussed. For more information about running functions in parallel, we gladly refer to the `parallel` manual and the document *How-to-go parallel in R*, written by Max Gordon. We thank Max Gordon for his previous work on this subject.

Time gain with small datasets

In Figure 1, the processing time is compared between `mice` and `parlMICE`. The simulated dataset has 1000 cases and 4 variables with covariance structure

$$\Sigma = \begin{array}{c c} & \begin{array}{cccc} V_1 & V_2 & V_3 & V_4 \end{array} \\ \begin{array}{c} V_1 \\ V_2 \\ V_3 \\ V_4 \end{array} & \begin{pmatrix} 1 & 0.5 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 & 0.5 \\ 0.5 & 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 0.5 & 1 \end{pmatrix} \end{array}.$$

A MCAR missingness mechanism is imposed on the data. In total, 80 percent of the cases (rows) has missingness on one variable. All variables have missing values.

Multiple imputation is performed with the default arguments of `mice`. The procedure with `parlMICE` is repeated for 1, 2 and 3 cores. The results are shown in Figure 1.

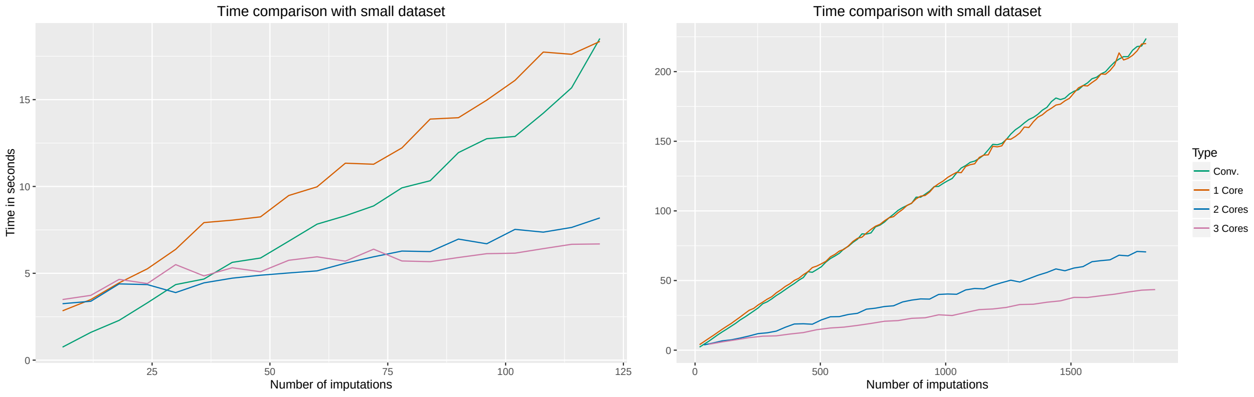

Figure 1. Processing time for small datasets. Multiple imputations are performed with `mice` (conventional) and wrapper function `parlMICE` (1, 2 and 3 cores respectively). The dataset has 1000 cases and 4 variables with a correlation of 0.5. 80 percent of the cases have one missing value based on MCAR missingness.

From Figure 1 it becomes apparent that for a small number of imputations, the conventional `mice` function works faster than the wrapper function. This is the case for `m` ≤ 35. From that point on, wrapper function `parlMICE` returns the imputations faster.

The right figure of Figure 1 shows the processing time for very high number of imputations. After the intersection, the lines follow the same trend: 3 cores perform the work faster than 2 cores, which work faster than 1 core. Naturally, using `parlMICE` with 1 core is similar to function `mice`, as the figure shows with the green and orange lines.

Time gain with large datasets

Another test was run with a (relatively) large dataset of 10000 cases and 8 variables. The covariance matrix was simply extended to the situation of 8 variables. In Figure 2, the results are shown.

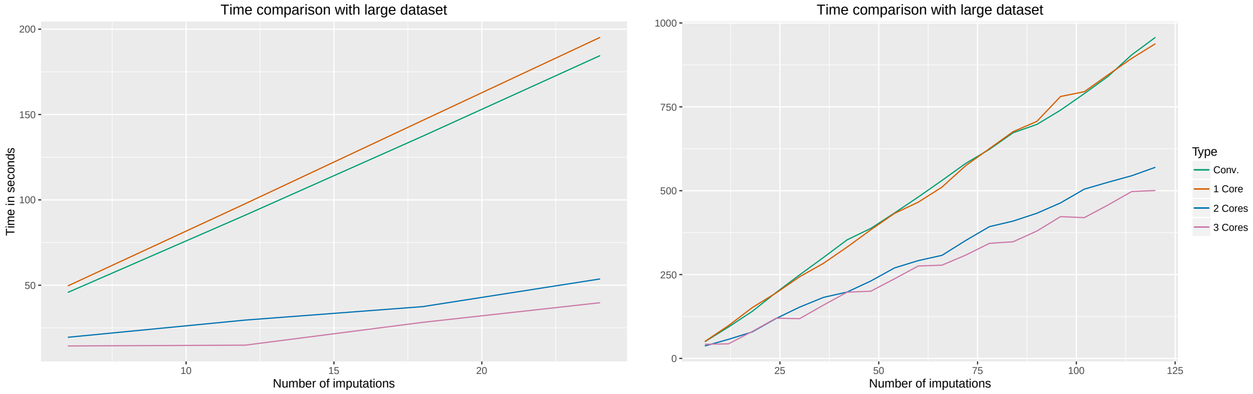

Figure 2. Processing time for large datasets. Multiple imputations are performed with `mice` (conventional) and wrapper function `parlMICE` (1, 2 and 3 cores respectively). The dataset has 10000 cases and 8 variables with a correlation of 0.5. 80 percent of the cases have one missing value based on MCAR missingness.

Figure 2 shows that for all imputations, function `parlMICE` works faster than `mice`. Even for very small number of imputations, running `mice` in parallel with `parlMICE` saves a significant amount of time. This time gain can easily reach 70 percent for 10 imputations and more.

There is not a huge difference between using 2 and 3 cores with wrapper function `parlMICE`. For all number of imputations, the procedure runs faster with 3 cores, even though the imputations have to be divided over the cores. It might therefore be desirable to use always as many cores as possible (but always keep 1!).

Default settings

We will now dig into the arguments of function `parlMICE`. Easy imputation of an incomplete dataset (say, `nhanes`) can be performed with `parlMICE` in the following way.

```
imp <- parlMICE(nhanes)
class(imp)
```

```
## [1] "mids"
```

The function returns a `mids` object as created by `mice`. In fact, `parlMICE` makes use of function `ibind` to combine the `mids` objects returned by the different cores. Therefore, the `call` of the `mids` object has slightly changed.

```
imp$call
```

```
## [[1]]
## mice(data = data, m = n.imp.core, printFlag = FALSE)
```

```
##
## [[2]]
## ibind(x = imp, y = imps[[i]])
##
## [[3]]
## ibind(x = imp, y = imps[[i]])
##
## [[4]]
## ibind(x = imp, y = imps[[i]])
##
## [[5]]
## ibind(x = imp, y = imps[[i]])
##
## [[6]]
## ibind(x = imp, y = imps[[i]])
##
## [[7]]
## ibind(x = imp, y = imps[[i]])
```

All other parts of the `mids` object are standard.

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 can be adjusted. For other possibilities with `mice`, we refer to the manual.

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

```
##    age    bmi    hyp    chl
## "norm" "norm" "norm" "norm"
```

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

Arguments `n.core`, `n.imp.core`

With `n.core`, the number of cores (or CPUs) is given. `n.imp.core` defines the number of imputations per core. Subsequently, the total number of imputations equals `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.

Subsequently, running the default `parlMICE` function on a computer with,

```r
detectCores()
```

```
## [1] 8
```

available, logical cores, results to `m = (4 - 1) * 2 = 6` imputations. We can check this by evaluating the `m` shown in the `mids` object.

```
imp$m
```

```
## [1] 14
```

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`:

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

Systems other than 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.

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

References

Gordon, M. (2015). How-to go parallel in R – basics + tips. Available at http://gforge.se/2015/02/how-to-go-parallel-in-r-basics-tips/

Manual base-package `Parallel`, available at https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf

Manual package `MICE`, available at https://cran.r-project.org/web/packages/mice/mice.pdf

Van Buuren, S., and C.G.M. Groothuis-Oudshoorn. 2011. "Mice: Multivariate Imputation by Chained Equations in R." *Journal of Statistical Software* 45 (3).