Introduction to the paglm package

Benjamin Christoffersen

2018-11-17

The motivation for the `parglm` package is a parallel version of the `glm` function. It solves the iteratively re-weighted least squares using a QR decomposition with column pivoting with `DGEQP3` function from LAPACK. The computation is done in parallel as in the `bam` function in the `mgcv` package. The cost is an additional $O(Mp^2 + p^3)$ where p is the number of coefficients and M is the number chunks to be computed in parallel. The advantage is that you do not need to compile the package with an optimized BLAS or LAPACK which support multithreading.

Example of computation time

Below, we perform estimate a logistic regression with 100000 observations and 50 covariates. We vary the number of cores being used with the `nthreads` argument to `parglm.control`.

```
#####
# simulate
n # number of observations
#> [1] 100000
p # number of covariates
#> [1] 50
set.seed(68024947)
X <- matrix(rnorm(n * p, 1/p, 1/sqrt(p)), n, ncol = p)
df <- data.frame(y = 1/(1 + exp(-(rowSums(X) - 1))) > runif(n), X)
```

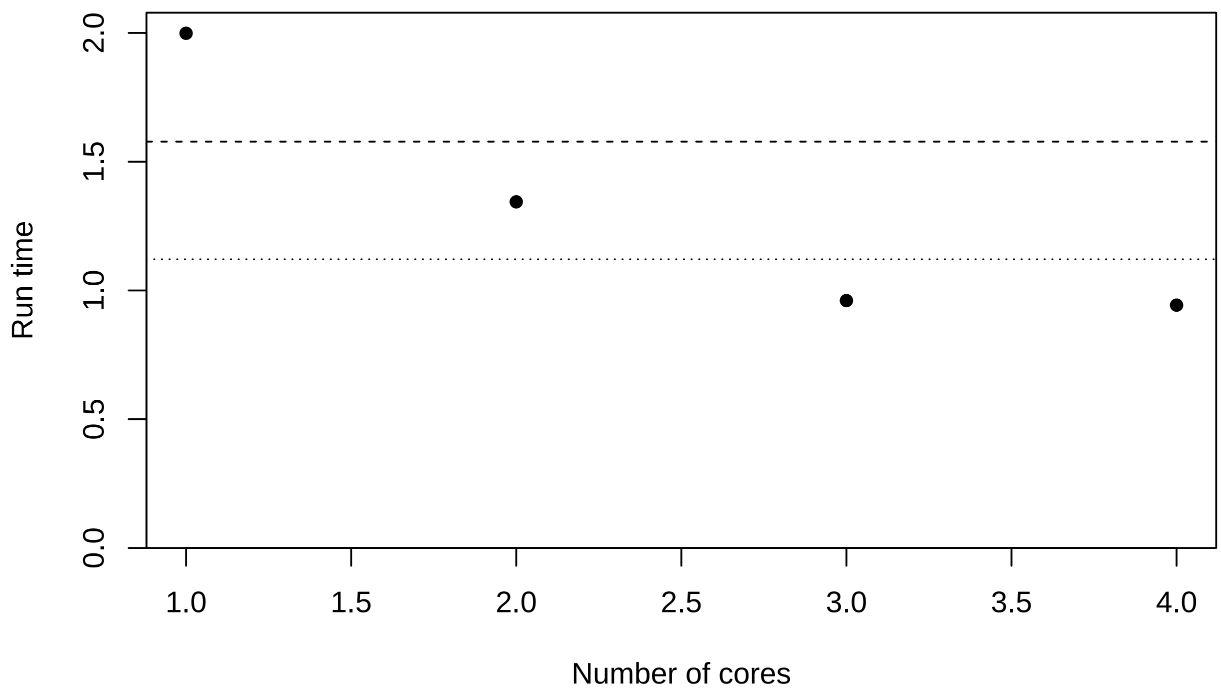
```
#####
# compute and measure time. Setup call to make
library(microbenchmark)
library(speedglm)
library(parglm)
cl <- list(
  quote(microbenchmark),
  glm      = quote(glm     (y ~ ., binomial(), df)),
  speedglm = quote(speedglm(y ~ ., family = binomial(), data = df)),
  times = 5L)
cl <- c(
  cl, lapply(1:n_threads, function(i) bquote(parglm(
      y ~ ., binomial(), df, control = parglm.control(nthreads = .(i))))))
names(cl)[5:(5L + n_threads - 1L)] <- paste0("parglm.", 1:n_threads)
cl <- as.call(cl)
cl # the call we make
#> microbenchmark(glm = glm(y ~ ., binomial(), df), speedglm = speedglm(y ~
#>     ., family = binomial(), data = df), times = 5L, parglm.1 = parglm(y ~
#>     ., binomial(), df, control = parglm.control(nthreads = 1L)),
#>     parglm.2 = parglm(y ~ ., binomial(), df, control = parglm.control(nthreads = 2L)),
#>     parglm.3 = parglm(y ~ ., binomial(), df, control = parglm.control(nthreads = 3L)),
#>     parglm.4 = parglm(y ~ ., binomial(), df, control = parglm.control(nthreads = 4L)))

out <- eval(cl)
out # result from `microbenchmark`
```

```
#> Unit: milliseconds
#>      expr       min        lq      mean    median        uq      max neval
#>       glm 1468.7372 1544.4974 1567.3155 1578.0583 1578.2989 1666.986     5
#>  speedglm 1101.9383 1116.9530 1139.4361 1121.0735 1165.7787 1191.437     5
#>  parglm.1 1853.6125 1913.5020 1975.3333 1998.7611 2031.8519 2078.939     5
#>  parglm.2 1158.4416 1297.6259 1326.3233 1344.1500 1415.3623 1416.037     5
#>  parglm.3  933.6888  938.0882  962.4052  960.5878  978.7986 1000.862     5
#>  parglm.4  855.1872  861.3530  938.1826  943.0131  980.7154 1050.644     5
```

The plot below shows median run times versus the number of cores. The dashed line is the median run time of `glm` and the dotted line is median run time of `speedglm`.

```
par(mar = c(4.5, 4.5, .5, .5))
o <- aggregate(time ~ expr, out, median)[, 2] / 10^9
ylim <- range(o, 0); ylim[2] <- ylim[2] + .04 * diff(ylim)
plot(1:n_threads, o[-(1:2)], xlab = "Number of cores", yaxs = "i",
     ylim = ylim, ylab = "Run time", pch = 16)
abline(h = o[1], lty = 2)
abline(h = o[2], lty = 3)
```



It is worth mentioning that `speedglm` computes the cross product of the weighted design matrix. This is advantages in terms of computation cost but may lead to unstable solutions. You can alter the number of observations in each parallel chunk with the `block_size` argument of `parglm.control`.

`parglm` does not at the moment handle close to singular problems as “neatly” as `glm` where `glm` forces some elements to be excluded. The single threaded performance of `parglm` is slower when there are more coefficients as seen above. The reason seems so be the `qr.qty` method in LAPACK, `dormqr`, which is slower then the LINPACK method, `dqrsl`, as illustrated below.

```
qr1 <- qr(X)
qr2 <- qr(X, LAPACK = TRUE)
microbenchmark::microbenchmark(
```

```r
  `qr LINPACK`     = qr(X),
  `qr LAPACK`      = qr(X, LAPACK = TRUE),
  `qr.qty LINPACK` = qr.qty(qr1, df$y),
  `qr.qty LAPACK`  = qr.qty(qr2, df$y),
  times = 25)
#> Unit: milliseconds
#>            expr       min        lq      mean    median        uq      max
#>      qr LINPACK 253.25373 269.28811 290.09839 280.64062 303.64468 370.6342
#>       qr LAPACK 259.49374 263.29502 280.69850 281.43785 291.18048 343.8328
#>  qr.qty LINPACK  22.66827  23.81078  32.19952  25.43528  30.39884 124.8084
#>   qr.qty LAPACK  89.83829  91.04047  95.66827  93.09084 100.86289 106.9401
#>  neval
#>     25
#>     25
#>     25
#>     25
```

Session info

```r
sessionInfo()
#> R version 3.5.0 (2018-04-23)
#> Platform: x86_64-w64-mingw32/x64 (64-bit)
#> Running under: Windows >= 8 x64 (build 9200)
#>
#> Matrix products: default
#>
#> locale:
#> [1] LC_COLLATE=English_United States.1252
#> [2] LC_CTYPE=English_United States.1252
#> [3] LC_MONETARY=English_United States.1252
#> [4] LC_NUMERIC=C
#> [5] LC_TIME=English_United States.1252
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base
#>
#> other attached packages:
#> [1] speedglm_0.3-2       MASS_7.3-49          Matrix_1.2-14
#> [4] microbenchmark_1.4-4 parglm_0.1.0
#>
#> loaded via a namespace (and not attached):
#>  [1] Rcpp_1.0.0                knitr_1.20
#>  [3] magrittr_1.5              devtools_1.13.6
#>  [5] lattice_0.20-35           RcppArmadillo_0.9.100.5.0
#>  [7] stringr_1.3.0             tools_3.5.0
#>  [9] grid_3.5.0                xfun_0.4
#> [11] tinytex_0.9               withr_2.1.2
#> [13] htmltools_0.3.6           yaml_2.1.18
#> [15] rprojroot_1.3-2           digest_0.6.15
#> [17] codetools_0.2-15          memoise_1.1.0
#> [19] evaluate_0.10.1           rmarkdown_1.9
#> [21] stringi_1.1.7             compiler_3.5.0
```

```
#> [23] backports_1.1.2
```