Multicore programming for embarrassingly parallel problems

Running a computation on more than one CPU of a multicore computer may increase its speed.

An embarrassingly parallel problem is one which is easy to break into sub-problems which can be solved in parallel. Examples of embarassingly parallel problems include:

- Simulations
- Resampling methods like the jackknife (327-2 hw4.Rmd) and bootstrap
- Calling optimization functions with many randomly-chosen initial values for parameters
- Cross-validation of models to correct for over-fitting
- Collecting simple statistics across a large data set partitioned into subsets

The `parallel` package gives access to multicore versions of `lapply()` and `mapply()` (and other functions, including more of the `apply` family) for embarrassingly parallel problems.

- `require("parallel")` loads the `parallel` package.
- `detectCores()` returns the number of CPU cores on your computer.
- Mac and Linux users:
  - `mclapply(X, FUN, ..., mc.cores=1)` ("multicore list apply") uses `mc.cores` cores (if available) to apply `FUN` to each element of vector or list `X`, returning a list
  - `mcmapply(FUN, ..., mc.cores=1)` ("multicore multiple arguments apply") uses `mc.cores` cores (if available) to apply `FUN` to all first elements of the several vectors in `...`, then to all second elements, etc.
- Windows users:
  - Prepare by calling
    `n.cores = detectCores()`
    `cluster = makePSOCKcluster(names=n.cores)`
    `clusterEvalQ(cl=cluster, expr) #` only if each core in cluster `cl` needs setup code in `expr`
  - Use `parLapply(cl=cluster, X, fun, ...)` instead of `mclapply()`
  - Use `clusterMap(cl=cluster, fun, ...)` instead of `mcmapply()`
  - Call `stopCluster(cl=cluster)` to clean up.
  - Let R through your firewall if asked

e.g. Here are two examples.

- See `nfl.R`

- See `mandelbrot.R`

For more, see `?parallel` and click the “Index” link at the bottom.