Automatic Parallelism Through R Code Analysis

Clark Fitzgerald

For Summer 2017 Research I plan to develop experimental software capable of converting serial R programs into parallel programs using multiprocessing. This automatic program tranformation differs from current parallel technologies which require the user to explicitly write code for a given parallel programming model. The software will have two major components: performance profiling, and programmatic transformation of the code into a parallel version.

Opportunities for parallelism can be found through analysis of base R's apply family of functions using the CodeDepends package [2]. The apply family includes `lapply, apply, sapply, tapply, by, mapply, Map, vapply, outer, by, replicate`. These are all variants of the map step of the map reduce computational model which has been successful for implementing large scale parallel systems [1].

Performance profiling measures how long the program spends executing each part of the code. The profiling together with estimates of the overhead associated with multiprocessing on the particular machine allow us to determine if it's actually worth it to parallelize a given R expression. If the overhead is larger than the speed gain then that expression should be left in serial form.

I will apply the software to several test cases including:

1. Basic algorithms operating on $n \times p$ matrices
2. Statistical simulations using `replicate()`
3. Practical data analysis on 100's of GB of traffic sensor data

To pass the tests the software should be capable of generating a parallel program that rivals the speed of a hand written parallel version. The practical data analysis use case will be the most challenging, because it's more difficult to profile, and the physical organization of data on disk imposes additional constraints. Other general challenges include respecting R's dynamic scoping rules, avoiding nested parallelism, and handling complex control flow.

I expect to produce a working prototype of the system that can handle basic algorithms and simulations by the end of the summer. This incorporates more intelligence into the system, freeing the user to write higher level code that addresses the specifics of their problem rather than the specifics of a particular parallel interface.

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[2] Duncan Temple Lang, Roger Peng, Deborah Nolan, and Gabriel Becker. *CodeDepends: Analysis of R code for reproducible research and code comprehension*, 2017. R package version 0.4-2.