HIGH PERFORMANCE COMPUTING

Applied Analytics: Frameworks and Methods 1

Outline

- Elements of R infrastructure that limit performance
- Profiling R Code
- Simple things to make R go fasteR
- Compile Code
- Compiled Languages
- Using GPUs
- Addressing constraint of RAM
- Parallel Processing
- Process on Database
- Distributed Processing

Write Code to Solve Problem First, Optimize Later

The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil (or at least most of it) in programming.

(Donald Knuth: https://en.wikiquote.org/wiki/Donald_Knuth)

Constraints to Speed

- Computing performance – CPU, RAM, I/O
- R is interpreted on the fly
- R is single threaded
- R requires all data to be loaded into memory
- Algorithm design affects time and space complexity

Profiling

- Speed Tests for Benchmarking
  - *Execution time: system.time()*
  - *Repeated time measurement: benchmark()*
  - *Distribution of execution time: microbenchmark()*
- Spotting Bottlenecks
  - *Rprof() (from utils)*
  - *Rstudio friendly chart: profvis({ })*
- Memory Utilization
  - *object.size(), pryr::object_size()*
  - *Rprof*
  - *profvis({ })*

Simple Things to make R go fasteR

- Better computer
- Vectorization (over loops)
- Use built-in functions
- Use faster functions
- Pre-allocate memory
- Use simpler data structures
- Use hash tables for frequent lookups on large data
- Use faster, more efficient packages

Compile Code

- R is an interpreted language which makes interactive programming possible.
- This places a greater burden on the computer which needs to translate R code into a machine understandable format.
- Lower level programming languages achieve better performance.
- Performance of R can be improved by compiling R code before execution
  - *library(compiler); cmpfun(); compile()*
  - *JIT Compiler: enableJIT(level=3)*

Compiled Languages

- The ultimate way to benefit from the power of compiled languages is to write code in C++.
- Need development tools for this
  - *For Windows install Rtools*
  - *For Mac, install the Xcode Command Line Tools.*
- R Packages that support writing C++ code
  - *library(inline)*
  - *library(Rcpp)*

Using GPUs

- GPU chips can speed up the execution of certain types of R code
- GPU programming
  - *CUDA for nVIDIA*
  - *OpenCL is brand agnostic*
- R Packages to leverage GPUs
  - *gputools*
  - *gmatrix*
  - *RCUDA*
  - OpenCL

Addressing Constraint of RAM

- Use RAM Wisely
  - *Use pointers instead of creating an identical copy of an object*
  - *Take out the trash*
  - *Calculate on the fly instead of storing*
  - *Move inactive data to disk*
- Use memory-efficient data structures
  - *Consider data type. E.g. complex>numeric>integer*
  - *Sparse matrices*
  - *Symmetric matrices*
  - *Bit vectors*
- Memory Mapped Files
  - *Store data on disk in the form of memory-mapped files and load the data into the memory for processing one small chunk at a time*

Parallel Processing

- In the last couple of years, computers have gotten faster but this is mostly attributable to multi-core processors that do parallel processing
- Unfortunately, R is single threaded
- Fortunately, there are now ways to do parallel computing, thereby leveraging the power of multiple cores.
- R programs can be written in order to run in parallel but the extent of parallelism depends on the computing task involved. It is easier for embarrassingly parallel tasks.
- Several R packages that allow code to be executed in parallel. E.g. library(parallel)

Process on Database

- For large databases or data that changes frequently, downloading the data into R may not be efficient or even feasible. One approach to addressing this issue is to move some computation to the database.
- Here are a few ways
  - *Computation with SQL. Packages that provides a database interface include RPostgreSQL, RMySQL, and ROracle.*
  - *For those who would rather work with R syntax, there are packages that can translate R syntax into SQL statements that are then executed on the database. These include dplyr and PivotalR.*
  - *Running advanced computation in the database using database specific algorithms or open source projects like MADlib.*
  - *Using columnar databases (e.g., MonetDB) for improved performance.*
  - *Using array databases (e.g., SciDB) for maximum scientific computing performance.*

Distributed Computing

- One of the solutions to analyzing large datasets is to use a distributed computing environment such as Hadoop.
- Apach Hadoop enables distributed processing of large datasets across clusters of commodity servers
- Apache Spark, a part of the Hadoop ecosystem, works particularly well for machine learning problems.
  - *Apache Spark is a fast and general engine for large-scale data processing*
  - *Multi-stage in-memory primitives provides performance up to 100 times faster for certain applications*
  - *Allows user programs to load data into a cluster's memory and query it repeatedly*
  - *Well-suited to machine learning*
  - *All the major cloud platforms - AWS, Google Cloud, Microsoft Azure - will rent and run a cluster. R packages for using Spark include SparkR and sparklyr.*

Summary

- In this module we discussed
  - *elements of R infrastructure that limit performance*
  - *profiling R Code*
  - *simple things to make R go fasteR*
  - *compiled code*
  - *compiled languages*
  - *GPUs*
  - *constraint of RAM*
  - *parallel processing*
  - *processing on database*
  - *distributed processing*