A Case Study of R Performance Analysis and Optimization

Ruizhu Huang
Texas Advanced Computing Center
University of Texas
Austin, Texas
rhuang@tacc.utexas.edu

Weijia Xu
Texas Advanced Computing Center
University of Texas
Austin, Texas
xwj@tacc.utexas.edu

Silvia Liverani
School of Mathematical Science
Queen Mary University of London
London, U.K.
liveranis@gmail.com

Dave Hiltbrand
Department of Biology and Marine Biology
University of North Carolina
Wilmington, North Carolina
hiltbrandd@uncw.edu

Ann E. Stapleton
Department of Biology and Marine Biology
University of North Carolina
Wilmington, North Carolina
stapletona@uncw.edu

ABSTRACT

Although R has become an analytic platform for many scientific domains, high performance has rarely been a trait of R. The inefficiency can come from the R programming specification itself or the interpreter environment implementation. Profiling and optimizing useful R code can not only directly benefit domain science researchers but also increase the efficiency of R code to run on high performance computing resources. We use envirotyping analysis as an example. This analysis considers both genetic information and environment conditions to understand how these factors affect crop yields through multi-dimensional data collected from fields and simulations. The analysis has the potential to improve breeding schemes for better global crop yield. A central tool used to support this analysis is an R package, "*PReMiuM: Dirichlet Process Bayesian Clustering, Profile Regression*", whose computational complexity increases as numbers of observations and features grow. The package is a useful tool for Bayesian clustering and inference with broad application potentials if computational bottlenecks can be overcome. In this paper, we detail our experiences on detecting the bottlenecks and optimizing its performance. We present a general workflow for investigating general performance issues such as execution time and memory usage to understand R program behavior and thus helping the optimization of the code. The workflow can be applied to other R applications. With the approach presented here, R users can easily identify inefficient code block, search for potential optimization solutions, and efficiently utilize high performance computing resources for scientific research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

PEARC'18, July 22-26, 2018, Pittsburgh, PA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6446-1/18/07…$15.00
https://doi.org/10.1145/3219104.3219156

CCS CONCEPTS

• **General and reference** → **Cross-computing tools and techniques** → Performance • **Mathematics of computing** → **Mathematical software** → Statistical software

KEYWORDS

R, software profiling, performance optimization, PReMiuM

ACM Reference format:
R. Huang, W. Xu, S. Liverani, D. Hiltbrand and A. Stapleton 2018. A Case Study of R Performance Analysis and Optimization. In *Proceedings of ACM Practice & Experience in Advanced Research Computing Conference 2018 (PEARC'18), Pittsburgh, PA, USA, July 2018,* 6 pages. https://doi.org/10.1145/3219104.3219156

1. INTRODUCTION

R is a programming environment with its own language specification and interpreter environment. R has interfaces that enable user to access R and utilize subroutines written with other programming languages. Profiling and optimizing useful R code can not only directly benefit domain science researcher but also allow R code to run more effectively on high performance computing resources. Although R [1] has become an analytic platform for many domain sciences, high performance has not been a strong feature of R [2], [3]. The inefficiency can come both from the programming specification itself and the interpreter environment implementation [4]. Even many packages available on the Comprehensive R Archive Network (CRAN) have the need for performance improvement on execution time or memory consumption when encountering computation intensive simulation and big data.

With thousands of lines of code in an R package and heterogeneous programming languages, identifying bottlenecks

is extremely difficult if no single profiling tool can easily track all computations. Optimization and improvement of R applications using high performance computing resources is an even harder task and requires additional knowledge and expertise. Currently, a freely available and easily applicable workflow for investigating general performance issues is not generally available to the R community.

In this paper, we consider an R application for a specific use case, Envirotyping. Envirotyping analysis considers both genetic information and environment conditions to understand how those factors affect crop yield [5]–[7]. Progress in crop yield improvement relies on the identification of genotypes better adapted to their production environment. The analysis consists of analyzing multi-dimensional data collected from fields as well as through computational modeling for parameter optimization and simulations to understand analysis power. This kind of analysis has the potential to improve breeding schemes to increase global crop yield.

A promising central tool to support envirotyping analysis is the R package, "*PReMiuM*: Dirichlet Process Bayesian Clustering, Profile Regression" [8], though the computational complexity of the package increases as the number of observations and features increase. If computational bottlenecks could be overcome, the package would also be useful for Bayesian clustering and inference for many types of data analysis. The package uses Markov Chain Monte Carlo (MCMC) methods for jointly fitting sub-models. Therefore, the overall computation is costly even for moderate data sets. For example, before the profiling and optimization, it took over 17 hours to process just over 4,530 observations with 7 variables, 1 discrete and 6 continuous. The long running time was an insurmountable barrier for academic research and industrial applications, especially with current trends toward rapidly expanding sample size and feature space.

In this paper, we detail our experiences when detecting the bottlenecks in the R package *PReMiuM* and optimizing its implementation. In Section 2, we provide some statistical background on profile regression and an overview of software profiling tools in R. Section 3 presents a general workflow for code profiling and optimization in R using the R package *PReMiuM* as an example. We conclude and briefly discuss ongoing works in Section 4.

2. BACKGROUND AND RELATED WORK

2.1 Software profiling with R

Software profiling is the analysis of the time and memory consumption on each line or a function of code during its execution. The development of efficient code relies on identification of key bottlenecks [9]. In interpreted languages (including R) a few lines can form major bottlenecks and strikingly slow down a program, especially with big datasets.

With the increasing popularity of machine learning, data mining and big data analytics, code profiling is becoming extremely prominent for efficient programming. For many domain science fields, one potential barrier to scientific advance is inefficient code, which is very common in many R packages [2]. Since many factors involved in affecting the execution time and memory usage are difficult to foresee, and the bottlenecks are very difficult to identify, especially within thousands of lines of code for an R package, a profiling tool is becoming absolutely indispensable in efficient programming.

We have reviewed eight major profiling tools for R. Our assessment considers five criteria including: profiling results presentation format; quality of trace back the function call history; resolution of code profiling; ability to profile memory usage along with CPU time; and how actively the tools are updated and maintained. Table 1 shows a comparison between the functionalities of eight major R profiling tools. The first column indicates if the package generates an HTML report. The second column indicates if the package generates some visual graphical output to show the results. The "Function nesting" column shows whether the package is able to display the function invocation history and hierarchy across the set of functions. The "Line profiling" column states whether the package provides information related with each of the lines in the code, not only functions. The "Memory profiling" column states whether the package shows results of profiling memory usage and finally, the "Latest update" column is an indication on how actively the tools are being maintained and developed [10]. Since *profvis* [11] package covers all functionalities and has an active development status, it was used for profiling our use case. The *profivs* tool has a convenient interactive graphical interface for visualizing profiling results. However, none of the packages in Table 1 offer insights into code that is implemented in languages other than R (e.g. C, C++, or Fortran), which is a serious limitation for all current R profiling tools.

Table 1: Overall of profiling tools in R

	HTML report	Graphical output	Function nesting	Line profiling	Memory profiling	Latest update
GUIProfiler [12]	yes	yes	yes	yes	no	8/23/15
summaryRprof [13]	no	no	no	yes	minimal	-----
proftable (Github) [14]	no	no	yes	yes	no	7/19/15
aprof [15]	no	yes	yes	yes	yes	12/14/17
proftools [16]	no	yes	yes	no	no	1/13/16
profr [17]	no	yes	yes	no	no	4/22/14
Lineprof *	yes	yes	yes	yes	yes	11/13/15
profvis [11]	yes	yes	yes	yes	yes	2/22/18

* lineprof is deprecated and replaced by profvis

2.2 Software profiling with C

Since many R packages involve some functions written in C/C++ language, and currently no profiling tools in R can evaluate performance, Intel® VTune amplifier [18] can be a complement to R profiling tools. It provides advanced profiling capabilities with a single, friendly analysis interface. It can be used to profile mixed language program such as Python, C++, C and Fortran, but it cannot provide line by line profiling for R programs such as the line profiling provided by *profvis*. The usefulness of Intel® VTune amplifier to R profiling lies in the top hotspots summary where optimizing these hotspots typically results in improving overall performance. The column of module in the summary provides general information on whether the C/C++ or R program components need improvement.

2.3 Profile Regression

Profile regression is one type of mixture model with regression used to comprehensively cope with highly correlated and multi-level data. Traditional regression analyses that attempt to fit main effects and interactions of increasing order to such data become quickly unwieldy and lose effectiveness. One way to deal with these difficulties is to adopt a more top-down point of view, where inference is based on clusters representing covariate patterns as opposed to individual risk factors and specific interactions. Partition and clustering methods are semi-parametric approaches that aim to discretize a multi-dimensional risk surface into cells, also called clusters, having similar risks.

These clusters aim to represent groups of individuals that share common characteristics, leading to similar risk profiles. Heterogeneity is thus broken down by augmenting the model with an underlying latent structure that partitions the observations into more homogeneous subgroups or clusters [8].

For the envirotyping studies, the combined effect of climate, soil and genotype variables on crop yield arise from complex mixtures consisting of highly correlated combinations of individual components. Profile regression is a statistical modeling framework that is inherently suitable for modeling highly correlated data. *PReMiuM* profiling can thus be used to examine patterns of genotype and weather variants and relate these patterns to crop yield. The overall approach is to cluster joint patterns of exposures using a flexible mixture of distributions.

3. Workflow Overview

In this section, we profile the execution time of functions in the R package *PReMiuM* following the workflow as shown in Figure 1, we identify the bottlenecks, and we provide solutions to optimize the performance. The runtime of top level functions can be profiled by the *proc.time()* function in R environment. After locating the time-consuming functions, the *profvis* R package can be used to identify bottleneck functions. To optimize these functions, improvement can be made by replacing with efficient functions or parallelizing independent processes with multicore or multi-node approaches [2], [19]–[21]. With comparison of different solutions via a speed-up test, the improved program can be further examined using the Intel® VTune amplifier to find out whether the remaining hotspots exist in R or C parts of the program. If hotspots still exist in R, additional R profiling is needed. Otherwise, the C/C++ program components in the R package should be optimized.

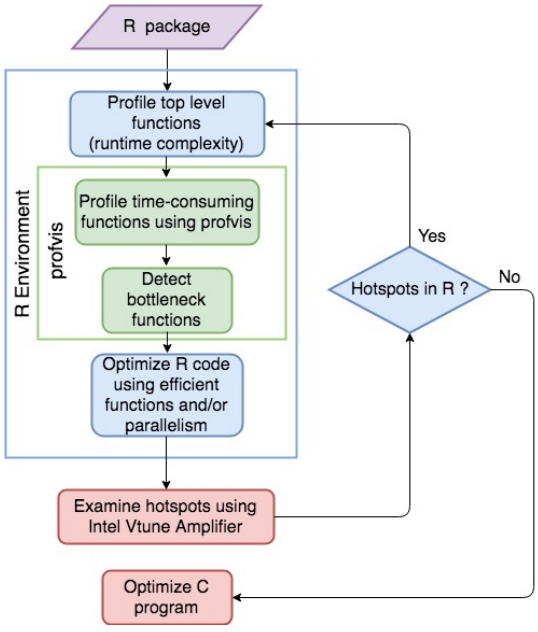

Figure 1: Workflow for R performance improvement

3.1 Testing Environment

All tests were conducted using Wrangler cluster at Texas Advanced Computing Center (TACC). Wrangler is a computer dedicated to run data intensive computing tasks. The Wrangler system at TACC includes 96 nodes. Each node has dual Intel Xeon E5 processors with 12 cores and 128GB memory. Each node has access to two file systems. One is a 10PB disk based Lustre file system. The other file system is a 0.5PB shared NAND flash storage system. All computing nodes are connected through one infiniband Mellanox switch with network transfer bandwidth up to 54Gbits/seconds as well as 40Gbits Ethernet. The flash storage system is directly accessible from each compute node through PCIe interface.

3.2 Functions and Dataset

To optimize the performance of the R package *PReMiuM* in solving envirotyping problems, we examined functions and the dataset features used in the analysis in order to detect bottlenecks in the source code.

3.2.1 Functions in the R package PReMiuM

In the preliminary investigation, a subset of implemented functions in *PReMiuM* package was used, including *profRegr* *calcDissimilarityMatrix*, *calcOptimalClustering*, *calcAvgRiskAndProfile* and *plotRiskProfile*. As shown in Figure 2, the post-processing functions have sequential dependency. The *profRegr* function is one of the core computational steps to fit the profile regression model. The current implementation is a subroutine implemented with C++ that can be invoked through R scripts. The output consists of a set of files containing traces and logs of the MCMC process. The results are used by the *calcDissimilarityMatrix* function to compute a dissimilarity matrix among all data objects. From the dissimilarity matrix, an optimal clustering assignment can be derived using the *calcOptimalClustering* function. The clustering results are then be used for post-result analyses to compute statistics and generate plots, two of which used in the envirotyping studies are *calcAvgRiskAndProfile* and *plotRiskProfile*. All functions except *profRegr* and *calcDissimilarityMatrix* are implemented using the R programming language. All functions are implemented as serial jobs with shared memory computing architecture.

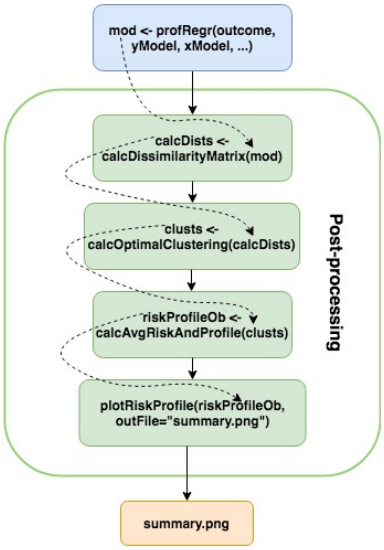

Figure 2: Function dependencies in *PReMiuM*

3.2.2 Dataset

The initial testing dataset for the envirotyping use case had 4,530 observations with outcome variable as yield, one factor covariate as brand-hybrid (847 levels) and 6 continuous weather variables:

air temperature (in Kelvin), wind (omega), air pressure, relative humidity, crosswind velocity, and vertical wind velocity.

3.3 Profiling Results

3.3.1 Top level profiling

Complexity of the sequential functions in *PReMiuM* (version 3.1.4) used in envirotyping research was tested by varying the number of observations and feature. These functions are also essential functions in *PReMiuM*, where the *profRegr* function fits a profile regression model and returns a number of files in the output directory and those output files are subsequently used by postprocessing functions including *calcDissimilarityMatrix*, *calcOptimalClustering*, *calcAvgRiskAndProfile* and *plotRiskProfile*. As shown in Figure 3, when 40, 80, 160, and 320 rows were randomly selected from the 4530 observations, the relative runtime to the baseline 40 rows of data increases with the number of observations and the *plotRiskProfile* function consumes almost 99% of the total runtime of the envirotyping analysis. In Figure 4, with an increasing simulated number of features and randomly selected 60 observations, the relative runtime for *calcAvgRiskAndProfile* and *plotRiskProfile* increase dramatically when increasing the number of covariates from 48 to 96, especially with larger numbers of clusters detected. The number of clusters detected varies with increasing number of rows and covariates and has substantial impact on the runtime. When the number of covariates becomes significantly larger than the number of clusters detected, the runtime of *calcAvgRiskAndProfile* grows faster than *plotRiskProfile*, because the *calcAvgRiskAndProfile* function computes the risks and profiles for all covariates and all clusters while *plotRiskProfile* only combines the profiles of the clusters for each covariate. For increasing number of continuous covariates, the most time consuming function shifts to *calcAvgRiskAndProfile*, but currently with only 6 continuous covariates for 4530 data points in the envirotyping dataset, the most urgent need was to identify the bottleneck within the *plotRiskProfile* function. Thus, the following code analysis focuses on this function.

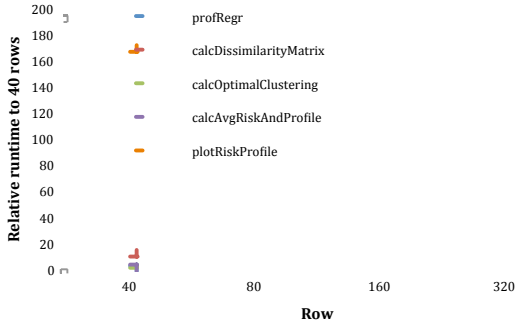

Figure 3: Complexity of top level functions by number of rows (baseline 40 rows)

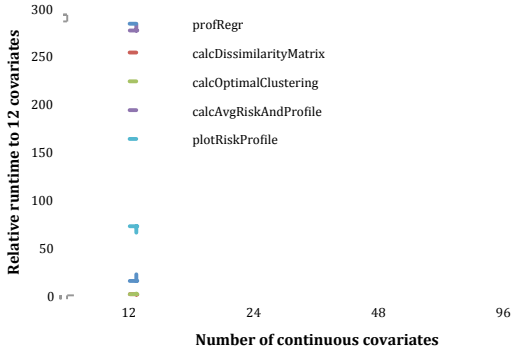

Figure 4: Complexity of top level functions by number of continuous covariates (baseline 12 covariates)

3.3.2 plotRiskProfile Function profiling

To examine the detailed runtime on each line of R code in *plotRiskProfile* function, we used a random sample of 60 observations in the full dataset. The reason for not using a full/big dataset for profiling is that the *profvis* package will run into out of memory issues when visualizing the profiling graph. A representative small sub-sample as a test set is sufficient and efficient in order to show which functions consume most of the time, especially when examining the code block with the major bottleneck.

The flame graph generated by *profvis* (Figure 5) shows the runtime by different levels of call stack in the *plotRiskProfile* function. The horizontal direction represents time in milliseconds, and the vertical direction represents the call stack. At the bottom-most items on the stack is the *plotRiskProfile* function being profiled. One level up on top of *plotRiskProfile*, when moving the censor to the cell, a popup window shows the label, total time, memory, aggregated total time and call stack depth. By examining second level of call stack, three functions, *rbind*, *rownames* and *print*, are called by the *plotRiskProfile* function. Note that for those functions that spent almost no time, information will not show up on the graph.

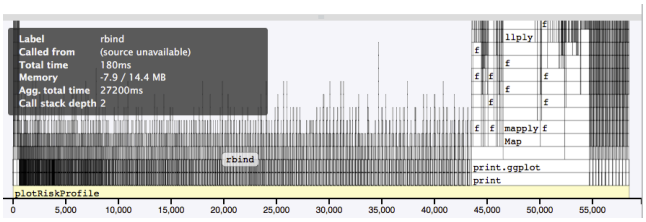

Figure 5: Flame graph of *profvis* result on *plotRiskProfile* function in *PReMiuM* package

The data view (Figure 6) shows detailed memory allocation and de-allocation. Interpreting this information can be misleading, because it does not necessarily reflect memory allocated and de-allocated at that line of code. The sampling profiler records information about memory allocations that happen between the previous sample and the current one. This means that the allocation/de-allocation values on that line may have actually occurred in a previous line of code [11].

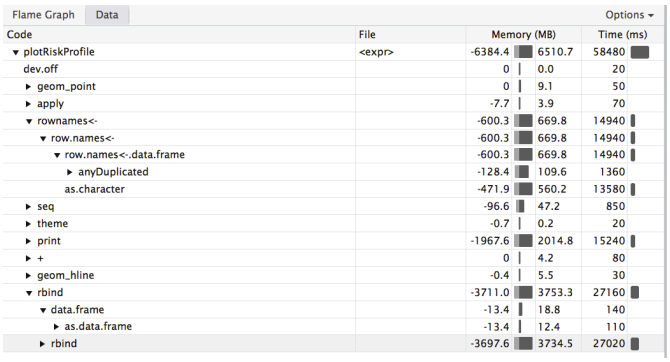

Figure 6: Date view of profvis result on plotRiskProfile function

3.4 Optimization of *plotRiskProfile* function

By looking at the *plotRiskProfile* in the Postprocess.R source code, we found 17 *rbind* and 17 *rownames* assignment function calls within for loops. They are the bottlenecks. These loops are used to combine outputs from previous post processing functions for plotting. The rownames assignment placed within for loops is unnecessary and can be done after completion of iterations.

The challenge here is dealing with combining data frames using the *rbind* function. Although *rbind* is a common and frequently used function in the R programming language, it has several drawbacks when encountering increasing volume of data: first, appending rows to a data frame using *rbind* is very slow because it fails to pre-allocate data structures. Second, each time we add a row, R needs to find a new contiguous block of memory to fit the data frame in, so the *rbind* operation leads to lots of copying. Third, the *rbind* also checks that the columns in the various data frames match by name. If they don't match, it will re-arrange them accordingly.

We found three solutions for optimizing the performance of the *plotRiskProfile* functions to address the drawbacks above:

Solution A. *do.call('rbind', list)*: Use *list <- vector('list', n)* to initialize and pre-allocate a predefined size of list to store dataframe and after the assignment of the list in the for loop, *do.call('rbind', myList)* can be used to execute the *rbind* function in order to combine a list of data frames all at once at the end.

Solution B. *rbindlist*: The *rbindlist(myList)* in the *data.table* [22] package can be used to efficiently append rows to a data table. The *data.table* is an enhanced version of *data.frame* and designed for fast operation of big data. It offers fast aggregation of large data (e.g. 100GB in RAM), fast ordered joins, fast add/modify/delete of columns by group using no copies at all, list columns, a fast friendly file reader and parallel file writer.

Solution C. *rbindlist* and *doMC*: The for loop used to combine the data frame can be parallelized by using *foreach* package [23] and *doMC* packages [24] in R. The *foreach* packge allows users to parallelize the for loop with minimal modification of their source code by adding *%dopar%* after the *foreach()* statement. The *doMC* package is a "parallel backend" for the *foreach* package. It provides a mechanism needed to execute *foreach* loops in parallel. The *foreach* package must be used in conjunction with a package such as *doMC* in order to execute code in parallel.

The pseudo code for each of the optimization solutions is listed in Table 2.

Table 2: Pseudo code of original and optimization solutions

Original
```
for ( c in myVector) {
    myDF<-rbind(myDF, data.frame("x1"=..., "x2"=...))
}
```

Solution A
```
myList <- vector('list', n)
for ( z in 1:n) {
    myList[[z]] <- data.frame(data.frame("x1"=..., "x2"=...))
}
myDF <- do.call('rbind', myList)
```

Solution B
```
library(data.table)
myList <- vector('list', n)
for ( z in 1:n) {
    myList[[z]] <- data.frame(data.frame("x1"=..., "x2"=...))
}
myDF <- rbindlist(myList)
```

Solution C
```
library(foreach)
library(doMC)
registerDoMC(detectCores()-2)
tmp = foreach(z = 1:n) %dopar% {
    data.frame(data.frame("x1"=..., "x2"=...))
}
myDF <- rbindlist(tmp)
```

The performance of the three optimization solutions developed for the *plotRiskProfile* function was measured by the speedup (runtime before optimization/runtime after optimization) as shown on Figure 7. Using Solution B or C for the full envirotyping dataset with 4530 rows of observation, optimization achieves runtime about 120 times faster than the original *rbind* within the for loop in the original *plotRiskProfile* function, dramatically reducing the runtime from 17 hours to less than 10 minutes. With increasing volume of data, the benefit of *rbindlist* is apparent as the speedup increases significantly compared to Solution A. It is worthwhile to note that Solution C performs a little better than Solution B with increasing rows of data because the *rbindlist* is not computation intensive in the for loop to be parallelized in *plotRishProfile*. The total runtime for the *plotRiskProfile* function includes other functions such as plotting which cannot be easily parallelized, so the parallel solution overall is not very appealing.

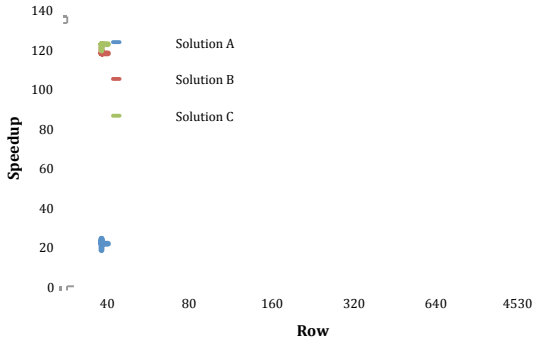

Figure 7: Performance of three optimization solutions to the plotRiskProfile function by sample size

After updating the *PReMiuM* package with Solution C, the Intel® VTune amplifier was used to examine the hotspots and show that top hotspots still exists in R program instead of within the C++ program, as shown in Figure 8. There are some copying operations from R module in *PReMiuM* that will need to improve.

Function	Module	CPU Time
__intel_avx_rep_memcpy	R	61.945s
RunGenCollect	libR.so	55.430s
bcEval	libR.so	19.045s
__printf_fp	libc-2.17.so	17.541s
Rf_findVarInFrame3	libR.so	15.536s
[Others]		346.813s

Figure 8: Hotspots detected by Intel VTune amplifier after optimization

4. SUMMARY AND ONGOING WORK

The performance improvement presented here make the promising new envirotyping analysis and modeling methods scalable and efficient and directly contribute to the development of relevant, publicly accessible known-truth genotype-environment simulations to allow improved breeding schemes for better global crop yield. The total runtime for the whole envirotypging analysis including modeling, postprocessing, and plotting has been significantly reduced from 17 hours to 17 minutes, by optimizing the R code of *plotRiskProfile* function in *PReMiuM* package. *PReMiuM* has been updated to include the improvements presented here and versioned as 3.1.6 on CRAN.

The method and process of profiling R code, detecting bottlenecks and testing solutions used in the envirotyping analysis can benefit the R user and developer communities. Most user developed programs or R packages on CRAN may perform well when dataset sizes are small, but large volumes of data could dramatically deteriorate the functionalities. Our example and explanation of how to profile and detect bottlenecks should be integrated as an essential part of testing in the R software development. Commonly used functions in R may not be the most efficient ones because of their implementations and specifications. For example, *rbind* is the most commonly used function in combining dataframes, but it lacks efficiency and should be replaced by *rbindlist* which is designed for fast operation and is available in the *data.table* package. Solutions to optimize R programs could be diverse and in the future there is a need to evaluate and compare under different circumstances using shared benchmarks.

Furthermore, many independent procedures can be parallelized via existing high performance R packages, which can be categorized into two types on the basis of the hardware requirement: single-node parallelism that requires multiple processing cores within a computer system and multi-node parallelism that requires access to computing clusters [2].

We are working on other functions required in the envirotyping analysis in *PReMiuM* package including *calcOptimalClustering* and *calcAvgRiskAndProfile* by following similar workflow described here. For the *profRegr* and *calcDissimilarityMatrix* functions written in C++, we do expect improvements by using parallelism solutions.

ACKNOWLEDGMENTS

This project was supported by National Research Initiative Competitive Grant no. 2017-67013-26188 from the USDA National Institute of Food and Agriculture and completed on the Wrangler cluster, which is generously funded by the National Science Foundation (NSF) through award #ACI-1447307. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

REFERENCES

[1] R Development Core Team, "R: A language and environment for statistical computing." Vienna, Austria, 2005.

[2] W. Xu, R. Huang, H. Zhang, Y. El-Khamra, and D. Walling, "Empowering R with high performance computing resources for big data analytics," in *Conquering Big Data with High Performance Computing*, 2016.

[3] R. Huang and W. Xu, "Performance Evaluation of Enabling Logistic Regression for Big Data with R," *2015 IEEE International Conference on Big Data*, 2015.

[4] H. Wickham, *Advanced R*. CRC Press, 2014.

[5] Y. Xu, "Envirotyping for deciphering environmental impacts on crop plants," *Theoretical and Applied Genetics*. 2016.

[6] M. Cooper *et al.*, "Predicting the future of plant breeding: complementing empirical evaluation with genetic prediction," *Crop and Pasture Science*, vol. 65, no. 4, pp. 311–336, 2014.

[7] K. Chenu, "Characterizing the crop environment–nature, significance and applications," in *Crop Physiology (Second Edition)*, Elsevier, 2015, pp. 321–348.

[8] S. Liverani, D. I. Hastie, L. Azizi, M. Papathomas, and S. Richardson, "PReMiuM: An R Package for Profile Regression Mixture Models Using Dirichlet Processes," *Journal of Statistical Software*, 2015.

[9] G. Wilson *et al.*, "Best practices for scientific computing," *PLoS Biology*, 2014.

[10] A. Rubio and F. de Villar, "Code Profiling in R: A Review of Existing Methods and an Introduction to Package GUIProfiler.," *R Journal*, vol. 7, no. 2, 2015.

[11] W. Chang and J. Luraschi, "profvis: Interactive visualizations for profiling R code [Software]." 2016.

[12] F. de Villar and A. Rubio, "GUIProfiler: Profiler Graphical User Interface," *R package version 0.1*, vol. 2, 2014.

[13] R Development Core Team, "Writing R extensions," *R Foundation for Statistical Computing*, 1999.

[14] N. Ross, "proftable." [Online]. Available: https://github.com/noamross/noamtools/blob/master/R/proftable.R. [Accessed: 26-Mar-2018].

[15] M. D. Visser, "aprof: Amdahl's Profiler, Directed Optimization Made Easy," *URL http://CRAN. R-project. org/package= aprof. R package version 0.2*, vol. 4, 2014.

[16] L. Tierney and R. Jarjour, "proftools: Profile Output Processing Tools for R," *R package*, p. 0, 2007.

[17] H. Wickham, "profr: An alternative display for profiling information," *R package. URL http://had. co. nz/profr*, 2008.

[18] Intel Developer Zone, "Intel VTune Amplifier, 2017," *Documentation at the URL: https://software. intel. com/en-us/intel-vtune-amplifier-xe-support/documentation.*

[19] F. C. Liu, W. Xu, M. Belgin, R. Huang, and B. C. Fleischer, "Insights into Research Computing Operations using Big Data-Powered Log Analysis," in *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact - PEARC17*, 2017.

[20] W. Xu, R. Huang, M. Esteva, J. Song, and R. Walls, "Content-based comparison for collections identification," in *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*, 2016.

[21] R. Huang, W. Xu, and R. McLay, "A web interface for XALT log data analysis," in *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, 2016, p. 31.

[22] M. Dowle, T. Short, S. Lianoglou, R. Saporta, A. Srinivasan, and E. Antonyan, "data. table: Extension of data. frame," 2014.

[23] Revolution Analytics and S. Weston, "foreach: Provides Foreach Looping Construct for R," *R package version*, vol. 1, no. 3, p. 1, 2015.

[24] Revolution Analytics, "doMC: Foreach parallel adaptor for the multicore package," *R package version*, vol. 1, no. 3, 2014.