GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar Report

Using R for High-Performance Data Analytics

Christopher L. Luebbers

MatrNr: 20676955

Supervisor: Matthias Eulert

Georg-August-Universität Göttingen
Institute of Computer Science

March 25, 2024

Abstract

This report investigates the use of R in high-performance data analytics (HPDA). It gives attention to the journey of the R programming language from the statistical analytical tool of its genesis into a robust conduit for high-performance computing (HPC). In addition, as data volumes increase and analyses become increasingly complex, it becomes compulsory to carry out these analytic processes at scale efficiently.

These challenges include effectively managing large datasets, optimizing computational speed, and negotiating the complexities of data analyses within R. The underlying issues indicate that R needs to be improved to surpass others in performance and functionality to meet HPDA's requirements.

Existing solutions primarily rely on the R package ecosystem. Much is still desired to make this solution an easy complement to the R statistical toolkit, particularly for performance improvement over base R or existing alternatives like Python.

The following provides some insights into advanced features and optimization with R, such as techniques on memory management, integration of GPU computing, and parallel processing capabilities—strategically using C++ to boost R's performance levels. Benchmarking exercises and a practical case study in computational biology will illustrate how these improvements are applied and performed in the HPDA tasks.

Evaluations indicate that R has become increasingly competitive in the HPC arena, courtesy of strategic enhancements in memory management, GPU computing, and parallel processing. For example, C++ integration in R has the potential to improve computationally demanding applications significantly. R is uniquely adapted to handling large genomic datasets and performing sophisticated statistical analysis in computational biology, work that general-purpose tools might not gracefully execute. Therefore, the results uphold the task-specific approach of choosing between R and Python regarding the value they bring to practice in HPDA.

Contents

1 Introduction

High-Performance Data Analytics (HPDA) refers to the critical junctures between data science and HPC to make valuable inferences from massive datasets with speed and efficiency. Nowadays, anything from health to finance, from environmental science to digital marketing, needs the ability to process and analyze big bulks of data at an incredible speed. The expected market size is growing accordingly [Mor24]. Some of the challenges in carrying out such data-intensive tasks include the management of the sheer volume of the data and the other intricacy in its analysis—all done and worked on in a timely, effective, and accurate manner.

One of the challenges within this field of HPDA is the fact that there are numerous tools. Still, none seem perfect—one that is relatively easy to utilize and has a critical mass regarding community support yet is effective in performing complex analysis. The web's big data is broadly divided into several categories: time series and non-time series data, large arrays, and tables, which together account for most of the data available.

R [The23] has been a subject of interest due to its origins and strengths in statistical analysis. This is coupled with an all-encompassing package ecosystem [WA22], making it a powerful tool for high-performance analytics. In contrast, R faces challenges, such as memory management, speed, and scalability, when the datasets or computation tasks involved are large and complex. Existing attempts to improve R performance offered a workaround solution to some challenges: using packages for parallel computation and big data technologies. Nevertheless, these solutions require users to navigate many packages and integration complexities. This has led to debates over the suitability of R in high-performance contexts, particularly when compared to Python.

This report critically reviews R's performance capabilities in high-performance settings in three specific areas: memory management, GPU computing, and parallel processing. Therefore, this study seeks to demonstrate the potential of using R in HPDA by leveraging R packages, including `data.table`, `bigmemory`, `sparklyr`, and `tensorflow`. A critical assessment made through a series of benchmarks and case studies, especially in biology, is where R's statistical capabilities with the specific packages provide unmitigated benefits over Python.

Contributions to this report include

- Identifying R's functional capabilities in enabling and supporting HPDA, such as parallelism and memory management functionality.
- Showcasing Practical Application: This is achieved through an example case study approach that describes real-world use-case implementation of HPDA using R, with a focus on areas where the strengths of R are unique and specific to the target.

After the introduction, this report examines R's role in HPDA and considers its strengths and weaknesses. Further discussion is centered on memory management and its ability to cope with big data, focusing on how R extends such capabilities through some packages. Afterward, this report explores GPU computing and parallel processing in R compared to Python and demonstrates how R is used in biological case studies. Finally, the report summarizes the findings and reflects on R's future in the HPDA landscape. The primary purpose of this report is to give an understanding of the capability of R to handle the challenges of both complexity and largeness that datasets may present.

2 R Fundamentals in Data Analysis

R is a powerful statistical tool for data manipulation, modeling, and graphical analysis. This section will highlight R's core aspects that make it a valuable tool for data analytics.

2.1 Data Handling Efficiency

R is powerful at data manipulation, with dozens of packages such as `dplyr` and `data.table` providing powerful, quick, and handy tools for working with data. The features associated with R data preparation include replacing missing values, transforming data types, and normalizing datasets. These methods ensure that the data is primed for accurate analysis. R helps easily flow through data preparation—filtering rows, selecting columns, and summarizing information—to reach the analytics steps easily. The latter maximizes Exploratory Data Analysis (EDA) in R, which helps an analyst discover newly acquired insights and guide them for further analysis.

See figure 4 for an example. It shows natality data, so data about birth in the United States [Dri]. The columns about birth anomalies are of interest. Each row is a birth, and those columns indicate whether there was a birth anomaly. We want a table to see how many anomalies were found each month. The code in listing 4 could be used to import this data using the packages `readr` and `dplyr`. These data manipulation tools enhance exploratory Data Analysis (EDA) in R, enabling analysts to uncover insights and inform subsequent analyses quickly. The resulting dataframe can be seen in table 3.

2.2 Statistical Computing Capabilities

R was designed with statistics at its core. It provides many functions for descriptive statistics, inferential statistics, and statistical models. It provides a comprehensive environment, from the simplest t-tests and chi-squared tests to the most complicated linear and non-linear modeling. With its depth in analytical capabilities, this syntax simplicity brings forward the appeal R has for statistical analysis. It is possible in R to create statistical models and have several tools for interpretation and analysis. R provides comprehensive summaries and diagnostic plots that help understand model behavior and accuracy. Data scientists can quickly build models to understand relationships between variables; even a beginner can leverage this knowledge. For example, creating a linear regression model in R is straightforward, as seen in listing 1. In this case, if applied to the dataset, variables can be interpreted, the model's validity can be tested, and predictions can be made. This level of accessibility and depth in modeling makes R a powerful tool, even for basic applications.

2.3 Visualization for Large-Scale Data

Effectively visualizing data is critical in communicating findings, and R excels in this domain. R truly shines in data visualization, turning complex data sets into compelling visual stories. `ggplot2` is a powerful and versatile package for creating high-quality graphs and charts. Along with other visualization packages in R, it makes presenting data more intuitive and impactful. Whether through histograms, scatter plots, or complex multi-layered graphics, R enables the creation of visual narratives that make data understandable and engaging.

```r
lm_example <- lm(Month ~ Ane + Men + Cya, data=data)
summary(lm_example)
#> Call:
#> lm(formula = Month ~ Ane + Men + Cya, data = data)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -4.3204 -2.8382 -0.6839  2.4580  5.1843
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) -6.42556   23.22043  -0.277    0.789
#> Ane          0.01341    0.29111   0.046    0.964
#> Men         -0.02629    0.26175  -0.100    0.922
#> Cya          0.07444    0.08687   0.857    0.416
#>
#> Residual standard error: 4.014 on 8 degrees of freedom
#> Multiple R-squared:  0.09846,        Adjusted R-squared:  -0.2396
#> F-statistic: 0.2913 on 3 and 8 DF,  p-value: 0.8307
```

Listing 1: Creating a simple linear model from the data in table 3 with basic R.

The `tidyr` package is used in listing 5 to transform the dataframe from a wide one with many columns to one with only three. All anomalies are together in one column, and all case numbers in another. The `ggplot2` package is used to specify the data and aesthetics y and color, and add a geom. The + sign is remarkable. It makes visualizations easy to use and reuse because `ggplot2` objects can be stored, and another layer can be added with +. More layers are added in listing 6 as `ggplot2` uses the grammar of graphics. The grammar of graphics requires a **data** layer and a **geom** function with **mappings**, as can be seen in listing 5. Possible additional layers can be seen in listing 6 and are listed here:

- statistics and position, like linear regression models
- coordinate function, like polar coordinates
- facet function
- scale function, e.g. logarithmic
- theme function

The benefits and importance of using R in high-performance analytics are pointed out in "Data Visualization Using R for Researchers Who Do Not Use R," an article by Nordmann et al. [Nor+22]. This tutorial introduces practically how to visualize data with R and insists on its benefits of reproducibility, transparency, and the possibility of many fully customizable options. Further, it insists on the "grammar of graphics" used as a basis for data visualizations for the `ggplot2` package. It demonstrates how R allows more intricate and informative visualizations than commonly used point-and-click

software. The authors argued that these visualization options improved the look of data displays and increased transparency about the data distributions underneath. The paper has introduced R as a powerful tool for applications in the analysis and visualization of data in a multiplicity of broad scientific fields to new researchers [Nor+22].

2.4 R integration with Diverse Data Ecosystems

A crucial aspect of R's functionality is its ability to interface seamlessly with various data sources. This includes traditional databases and diverse data formats. R's packages like `DBI` and `odbc` [Jim23] allow for direct database interaction, enabling data retrieval and manipulation within the R environment. Moreover, R's compatibility with various data formats, e.g., CSV, Excel, JSON, and XML, demonstrates its adaptability. These features make it flexible due to its integration into several workflows.

2.5 Conclusion of Basic Applications

We have seen how R is a foundational tool in data analytics. Beginning with essential data manipulation, R simplifies cleaning and preparing data for analysis. Later, we looked at R's capabilities in statistical analysis and hypothesis testing, bringing up a colorful highlight as a statistical giant applying to simple and complex problems. This showcased the prowess of R in visualizing data to convert data into visual stories rich in insight using `ggplot2` and other packages. Lastly, R's flexibility in interfacing with varied data sources and formats underlines its ability to interface with various data analytic workflows.

3 Optimizing R for High-Performance Computing

Now, we will transition from R's basic applications to its role in HPC environments, exploring the advanced features that make R a viable option for HPDA tasks. This section highlights these key areas: memory management, GPU, parallel computing, benchmarking, and leveraging C++.

3.1 Efficient Memory Utilization

An essential aspect of high-performance settings is memory management. R inherently stores objects in physical memory, which, while ensuring fast access, can be a challenge with big data. R provides various tools and strategies to manage this effectively. For instance, more efficient data structures, like `data.table`, can significantly reduce memory overhead. There are techniques like memory mapping and databases for handling data that are too large to fit into memory. These methods allow R to process larger datasets than the physical memory would typically permit. It is also good practice to remove all unnecessary objects from memory and use packages designed for memory efficiency. Tools like the `bigmemory` [Mic13] package offer solutions for managing massive datasets in R.

When handling big data in R, a key player is `sparklyr` [Kal22]. `sparklyr` is a package that allows R users to leverage the power of Apache Spark. With `sparklyr`, R users can connect to a Spark cluster to leverage its ability to use the computation power of

distributed computing to scale up data analysis and processing. `sparklyr` provides R users an interface to work with Spark's distributed data frames, much like `dplyr`, performing many operations on filtering, summarizing, and querying big data sets. It also allows for executing machine learning algorithms and other complex data operations. The idea is to push as much work as possible to Spark and collect only what is needed. This is the integration of R with Spark, so data scientists can solve their big data problems right inside the R environment without leaving it.

3.2 GPU Computing

Another significant development is the integration of GPU computing. Unlike traditional CPUs, GPUs can dramatically accelerate computational tasks, especially parallelizable ones. Packages like `gputools` and the integration with TensorFlow through `tensorflow` package allow R to leverage the parallel processing power of GPUs for high-speed computations and machine learning tasks. This synergy between R and TensorFlow enables users to tackle complex tasks quickly and efficiently.

Most packages focus on using NVIDIA CUDA for GPU computing. The new `clrng` [Xu] package can use GPU technology independently of the vendor, which leads to substantial performance improvements when generating random numbers. We want to check for a significant association between those variables using the previous data in table 3. This is done using something called Fisher's exact test. It does many Monte Carlo simulations, which need to generate much random data. This kind of simulation is used in climate modeling, particle physics, pharmaceutical research, and everything else that involves simulations with significant uncertainties. Listing 2 shows how it is done in basic R by specifying the data and running a Monte Carlo simulation for 49 seconds.

```
1 time_cpu <- system.time(result_cpu <- stats::fisher.test(month,
2                                                          simulate.p.value = T\textsf{R}UE,
3                                                          B = 1015808))
```

Listing 2: Running a fisher test in basic R in 49 seconds [Xu].

To use the new `clrng` package, specify the stream of random numbers on the GPU, move the data to the GPU as a matrix, and run the simulation (see listing 3). It runs for 2.2 seconds.

A remarkable difference in GPU computations is noticeable in table 1. The same calculations are executed significantly faster thanks to GPUs' parallel processing capabilities.

Table 1: Comparison of Fisher's test simulation on different Devices [Xu].

Device	runtime	p-value
Intel 2.5 ghz	49.3	0.403804
AMD Radeon VIII	2.2	0.403507

```
library(clrng)
streams <- createStreamsGpu(n = 256*64)
month_gpu <- vclMatrix(month, type = "integer")
time_gpu <- system.time(result_gpu <- clrng::fisher.sim(month_gpu, 1e6,
                                                        streams=streams,
                                                        type= "double",
                                                        returnStatistics=T\textsf{R}UE,
                                                        Nglobal = c(256,64)))
```

Listing 3: Running a fisher test with `clrng` in 2.2 seconds [Xu].

3.3 Parallel Processing

GPU computing showed the potential of parallelization. Now, we move to how R can be used for parallel computing in general. Parallel computing, which involves processing multiple computations simultaneously using multiple processors, is crucial in HPC for efficiently handling big data analysis and complex computational tasks. Parallel computing techniques in the R programming environment are increasingly essential for accelerating data analytics processes and scientific research. R is not naturally parallel; however, it has grown to have robust parallel computing capabilities [MW11]. This allows for significant performance improvements, especially when dealing with big data processing.

Parallel computing can be implemented in R in several ways [23]. The different parallelization schemas are visualized in figure 1. One common approach is to handle embarrassingly parallel tasks. Those tasks are independent. These tasks can be performed independently without the need for communication between processors. Worker queues distribute tasks across a set of worker processes. They have a primary process that coordinates several R processes on different compute nodes using a shared file system. Shared memory parallelization splits tasks across multiple cores of a single machine. Another concept is message passing, which is used in distributed computing. It involves multiple processors communicating with each other over a network, each handling a part of the computation.

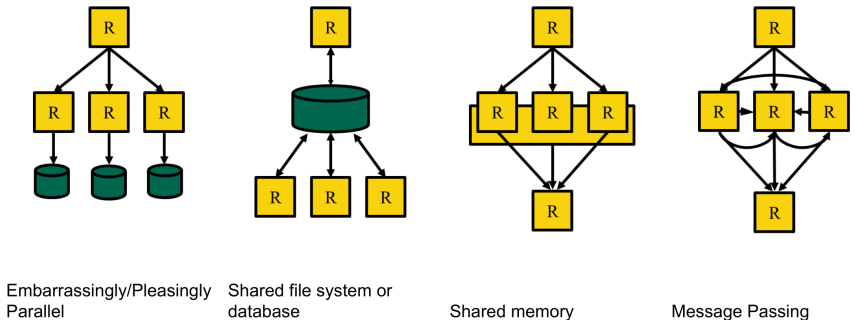


Figure 1: Parallelization Schemas in R [23].

R offers several packages for parallel computing. `foreach` [Fol23] and `doParallel` are widely used for loop-like parallel executions. They allow for easy parallel processing implementation without heavily modifying the code structure. The `parallel` package, included in R, provides parallel versions of the apply functions, such as `lapply` and `mapply`.

It is a straightforward way to start parallel processing, especially for those familiar with these functions. Packages like `future` [Ben21] and `promises` are used for more advanced parallelization.

Dirk Eddelbuettel's review on parallel computing with R reviews different approaches to parallelism that can be used in conjunction with R. These approaches range from compiler-level solutions like OpenMP and Intel TBB to process-parallel and message-passing parallelism and extend to leveraging big data technologies like Spark, Docker, and Kubernetes. Eddelbuettel highlights the `future` package in R, which integrates many of these parallel computing approaches, offering a versatile framework for parallel data processing [Edd19].

3.4 Benchmarking

Benchmarking in R provides one of the most necessary aspects to study code performance, especially in HPDA, since efficiency and speed are primary prerequisites. Benchmarking is a set of metrics that shows how good a given piece of code is likely to perform if placed in a production environment. Standard metrics used in HPDA performance evaluation include:

- **Execution time** is the time it takes a code segment to run, often measured in milliseconds or microseconds for high-performance tasks. It is the most direct performance measure. Lower times have a better efficient translation.
- **Memory Usage** is the amount of memory allocated at runtime by the code. Effective memory use is needed in HPDA to avoid establishing bottlenecks when processing large datasets.
- **CPU Utilization** shows the percentage of the CPU capacity used to execute a given task. CPU resources should efficiently be utilized for optimal code without undue strain or bottleneck.
- **Iterations per Second** indicates how often the execution of a given piece of code can occur within one second. It measures the responsiveness of the code's ability to do the task with repeatability.

The book "Advanced R" has a separate section on benchmarking [Wic19]. The `rprof` package provides a profiler that does stack sampling at regular intervals and gives you a general overview of where time is up during execution. This resolution could be higher, and vast amounts of profiling data can become unwieldy in complex code. For a more user-friendly interface, `profvis` visually represents profiling data as shown in figure 2.

It renders an HTML document where users can interactively drill down the performance metrics. It enables the developer to decompose visually the execution time among different functions and point to bottlenecks more intuitively when profiling with `profvis`. This generated flame graph represents the call stack with time spent in each function, thus quickly showing the visual difference across code segments. Another sophisticated R benchmarking tool is the `bench` package. It uses high-precision timers to measure the execution time of code blocks with more precision than `rprof`. Executions are then carried out to build the required total run time, which must be at least 0.5 seconds, to reduce the variability introduced by shorter test runs. The `bench` output is displayed with the corresponding minimum and median execution times, iterations per second, and

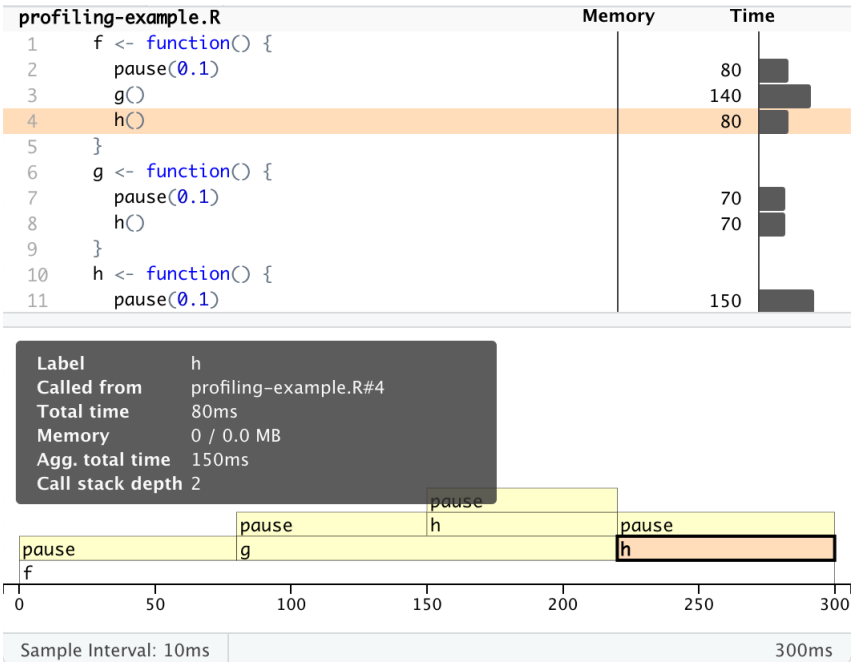


Figure 2: Benchmarking with `profvis` [Wic19].

the allocated memory usage during execution. An example can be seen in listing 7. It presents an invaluable treasure of granular data in R code optimization and the demanding requirements of HPDA.

3.5 Leveraging C++ to Enhance R Functions

One great way is to use C++ to soup up R functions to improve performance for the highly computationally intensive tasks typical of HPDA. This provides a seamless interface of R to C++, allowing the developer to write high-performance code without giving away the perks of a quick-and-dirty, interactive, highly readable prototyping environment. It speeds up the execution of the most critical code segments. Critical deadlines in an HPDA environment are often milliseconds long, so performance gains from C++ can be huge. `Rcpp` makes this available by:

- **Minimal Overhead:** With `Rcpp`, the overhead of calling C++ from R is minimal, and therefore, even small performance gains make C++ viable.
- **Cleanly written code:** `Rcpp` has a clean and expressive syntax that allows writing C++ code that is simple and readable, reducing the learning curve for R users already familiar with the concepts in C++.
- **Memory Management:** It automates data transfer between R and C++, handling complex memory management tasks.
- **Vectorization:** Vectorized operations can naturally be performed within `Rcpp` since, at the core, they are very efficiently supported in C++ the same way R does.

With `Rcpp`, users write parts of the code that are performance-critical in C++ and get it compiled and linked to R. The functions become callable, just like any other R

function. This is most useful in those scenarios where R's built-in functions inherently limit execution speed or limit them due to memory usage. A typical use would be reimplementing a computationally expensive loop in C++ to take advantage of the higher execution speed and more conscious memory management. This is often an excellent opportunity for significant performance improvements since C++ code may execute faster than its R counterpart. Such enhancements may speed up computation during extensive data processing or simulation runs with large iteration counts. This impact is performed quantitatively by benchmarking the original R function and the new R/C++ hybrid function. Better performance is generally observable in most metrics, from execution time to memory usage, making a compelling case for using `Rcpp` in performance-critical applications. Listing 7 shows an example of rewriting a loop with vector input and scalar output. The benchmark shows the C loop is more than 10 times faster than the custom R code. The highly optimized integrated `sum` function in R is on par with C++.

3.6 Advancing R's Computational Power

This section emphasized features and approaches essential to strengthen further the use of R for HPDA. The integration with Apache Spark allows R to work with datasets whose size exceeds physical memory limits. R effectively manages large datasets with memory management techniques. R with GPU computing offers an effortless, parallel processing architecture for significant data processing in some of the most complex tasks in simulation and data analysis, with timely results. Parallel computing is an approach that significantly enhances R's capabilities in handling big data analytics. By distributing tasks across multiple processors, greater efficiency and speed are achieved. Various parallelization techniques were explored: embarrassingly parallel tasks for independent operations, worker queues for task distribution, and message passing for distributed computing. Each technique offers unique advantages depending on the nature of the data and computational tasks. Several packages enhance the power of R in parallel computing. `foreach` and `doParallel` simplify loop-like operations, while the `parallel` package provides parallel versions of common apply functions. For more complex asynchronous tasks, `future` and `promises` offer greater flexibility. Benchmarking ensures that performance improvements are not claimed but are empirically evident. Embedding C++ code into the R environment gives both a prime role in hiking R as a tool for HPDA, combining flexibility and performance. This confluence of memory efficiency, GPU computing, parallel processing, benchmarking rigor, and C++ integration cement R as a statistical programming language and a robust tool for HPDA.

We are moving from enhancements in R for HPC to their practical application. The exploration now delves into specialized domains in which R's capabilities are helpful and critical.

4 Specialized Applications and Comparative Analysis

4.1 Case Study: R in Computational Biology

R's statistical strengths and comprehensive package ecosystem make it particularly well-suited for fields like biology, where data analysis and statistical modeling are critical. High-performance R packages enable biologists and bioinformaticians to process large genomic datasets, perform complex simulations, and conduct advanced statistical analyses previously unfeasible due to computational constraints. These capabilities allow for more sophisticated research methodologies and the potential for discoveries in genetics, evolutionary biology, and ecosystem modeling. R is often used in bioinformatics, pharmaceuticals, and genetic data.

In the following example [Met23], fruit flies have around 20.000 annotated genes. The question is if there is a correlation between those genes and a genetic disorder of the eyes that causes vision loss. `doParallel` and the `foreach` package are used. The benchmark is shown in figure 3. The y-axis shows the execution time in seconds. The calculations are benchmarked for the subsample, with the x-axis values 18, 20, and 22 representing the study's complexity levels. The number of processes was initiated and systematically increased until the perfect count was found, in which computation efficiency attained peak value. The optimal process count is arising for computation with higher complexity. Communication increases by the number of processes. With an increase in the number of processes, performance may stay the same. This is to say that computational optimization has to involve customization simultaneously with understanding the context and the environment in the software and hardware being used.

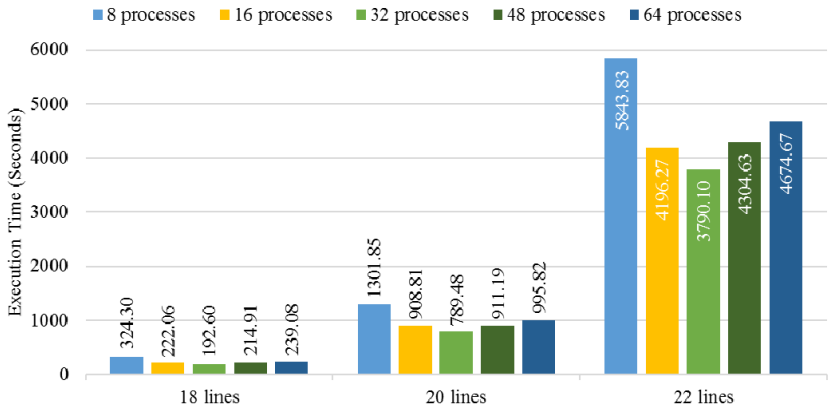


Figure 3: Benchmark of subsamples [Met23]. There is an optimal minium in execution time for 32 processes.

Ioannis Charalampopoulos presented the paper "The R Language as a Tool for Biometeorological Research" [Cha20], which discusses using R in biometeorology, an interdisciplinary area between biology and meteorology. As one of the open-source programming languages, the profile of R has become central to geosciences and related research fields for the last twenty years in research involving diverse datasets encompassing atmospheric conditions and biological agents like plants, animals, and man. The flexibility and capacity of

automation in the analysis processes, from big data takeover to supporting reproducibility, have made R the leading choice in academia. Charalampopoulos elaborates on the structure of the R workflow in the context of the special needs related to biometeorological research. He thoroughly describes some of the most valuable and popular R software packages and treats the field's particular requirements. This paper attempts to introduce R to biometeorologists, who propose, based on R's capacities, a roadmap of how scientific work may be collaboratively undertaken [Cha20].

4.2 R vs. Python: A Comparative Performance Review

In the context of HPDA, R is always compared with Python since it has long been reputed to be capable of handling computationally intensive operations. Performance depends mainly on the nature of the workloads executed in each language and the libraries in use.

With its rich and immense ecosystem and support for parallel and GPU computing, this is where Python most often becomes the best candidate for such general-purpose tasks. In contrast, its high-end specialized packages, such as `Rcpp` and `data.table`, and parallel computing suites tools like `parallel`, `foreach`, `doParallel`, and `Rmpi` have done much to close the gap in HPC contexts [Cha19]. These packages increase R's capability to carry out data-intensive tasks and hence widely improve its popularity in HPC applications.

Recent studies have comprehensively reviewed performing tasks such as data processing using R, Python, and Rust [Bei+23]. The results of this study show that although every other language can perform those tasks, the difference is significant. The tests show that Python and Rust are just about the best, while the execution time to complete the same task is much longer in the R language, see table 2. This fact underlines the benefit of Python's versatility and places it between the most powerful languages for GPU tasks, entering it into competition in many HPDA applications [Bei+23].

Table 2: Processing 65 GB in 586 `JSON` files in different setups. R is significantly slower in this specific use case [Bei+23]. Note that neither code was optimized.

Variant	Runtime[HH:mm:ss.SS]
pypy3_7	00 : 11 : 55.14
pypy3_8	00 : 11 : 13.73
pypy3_9	00 : 10 : 32.92
python3_7	00 : 18 : 55.79
python3_8	00 : 19 : 10.60
python3_9	00 : 19 : 54.83
python3_10	00 : 17 : 02.74
python3_11	00 : 16 : 00.72
r_rjson	03 : 00 : 56.25
rust_serde	00 : 04 : 30.78
rust_serde_rayon_2thread	00 : 02 : 48.65
rust_serde_rayon_3thread	00 : 02 : 04.94
rust_serde_rayon_4thread	00 : 01 : 45.59

R's significant statistical computation and bioinformatics make it the best tool for domains that require complex statistical analysis. That debate continues—not over a clear

leader being declared but over exactly which context each language excels in. Comparative studies have often highlighted scenarios where one language may outstrip the other regarding efficiency. Ultimately, the job requirement against which HPDA is deployed is considered: the complexity of calculations made and available hardware resources. By its very nature, this should ensure that the language chosen is commensurate with the goals and constraints of the given HPDA project.

5 Conclusion and Future Directions

Conclusion As we conclude, let us reflect on the journey through HPDA using R. Using R was evidenced to show a variety of flexibility in data analysis and visualization of the language and capabilities of performing some basic statistical methods. This was followed by packages that helped transit to the advanced level of applications that encompass GPU computations, including application packages like `clrng` and `tensorflow`. This demonstrated how R could be adapted for high-performance tasks, significantly enhancing computational speed. Further insight into the depth of R's capabilities in parallel computing was achieved by investigating the variety of techniques and packages of `foreach`, `doParallel`, and `future` that enabled R to control the parallelism of big data more efficiently. This, coupled with its high-end memory management, GPU computing, big data processing capabilities, and robust statistical analysis features, is a forceful tool in the HPDA landscape. It refers to a setting dealing with software for big data analysis, including value provision in many research and industry applications. While Python is still prevalent in data analytics because of its general-purpose ability and many libraries, R is preferred for a few specific cases requiring solid statistical analysis and graphical representations. These have ensured the dominance of both R and Python for HPDA. However, the choice will largely be dictated by the task's exact requirements in terms of the type of data, nature of computational resources, and desired analysis outcomes.

Key Takeaways

- **Memory Management:** Advances in memory management within R, such as `data.table` and `bigmemory`, address scalability issues
- **GPU Computing:** The integration of GPU computing into the R ecosystem offers substantial performance gains for suitable tasks
- **Parallel Processing:** R's capabilities for parallel computing have expanded, with packages like `parallel`, `foreach`, and `future` facilitating more efficient data processing across multiple cores and nodes.
- **Applications in Biology:** R's application in biological research underscores its versatility, enabling intricate analyses across biometeorology, genetics, and biodiversity conservation.

R's Evolving Role in HPDA and Beyond Future developments in R and its package ecosystem will likely enhance its utility in HPDA. Several challenges and considerations remain. Addressing these is crucial for R's continued growth and effectiveness in HPC environments. Looking ahead, several avenues for further research and development could enhance R's utility in HPDA:

- **Improving Scalability:** Continued efforts to enhance R's ability to handle large datasets efficiently, particularly in distributed computing environments.
- **GPU Computing Enhancements:** Expansion of R's GPU computing capabilities, including broader support for different GPU architectures and more intuitive data science interfaces.
- **Interoperability with Other Tools:** Further development of interoperability features between R and other data analytics tools and platforms to leverage the strengths of each in a complementary manner.
- **Educational Resources:** Increased availability of educational materials and resources focused on using R in HPDA to lower the entry barrier for new users and facilitate sharing best practices within the community.

In conclusion, R remains a powerful tool for HPDA capable of addressing complex data analysis challenges across various domains. As computational technologies evolve, so will R and its ecosystem, further solidifying its role in the data science and analytics landscape.

References

[23] *AI Training Series: High Performance Data Analytics*. Leibniz Supercomputing Centre, May 2023. (Visited on 01/11/2024).

[Bei+23] Lukas Beierlieb et al. "Efficient Data Processing: Assessing the Performance of Different Programming Languages". In: *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering* (2023). DOI: 10.1145/3578245.3584691.

[Ben21] Henrik Bengtsson. "A Unifying Framework for Parallel and Distributed Processing in R using Futures". In: *The R Journal* 13.2 (2021), pp. 208–227. DOI: 10.32614/RJ-2021-048. URL: https://doi.org/10.32614/RJ-2021-048.

[Cha19] Rahim K. Charania. "Exploring and Benchmarking High Performance & Scientific Computing using R R HPC Packages and Lower level compiled languages A Comparative Study". In: *ArXiv* abs/1904.03343 (2019). DOI: 10.13140/RG.2.2.16143.43680.

[Cha20] Ioannis Charalampopoulos. "The R Language as a Tool for Biometeorological Research". In: *Atmosphere* (2020). DOI: 10.3390/atmos11070682.

[Dri] Anne (CDC/OPHSS/NCHS) Driscoll. "User Guide to the 2018 Natality Public Use File". In: (). URL: http://www.cdc.gov/nchs/data_access/VitalStatsOnline.htm.

[Edd19] Dirk Eddelbuettel. "Parallel computing with R: A brief review". In: *Wiley Interdisciplinary Reviews: Computational Statistics* 13 (2019). DOI: 10.1002/wics.1515.

[Fol23] Steve Weston Folashade Daniel. *foreach: Provides Foreach Looping Construct*. 2023. URL: https://github.com/RevolutionAnalytics/foreach (visited on 01/15/2024).

[Jim23] Hadley Wickham Jim Hester. *odbc: Connect to ODBC Compatible Databases (using the DBI Interface)*. 2023. URL: https://github.com/r-dbi/odbc (visited on 01/15/2024).

[Kal22] Tomasz Kalinowski. *Posit AI Blog: Deep Learning with R, 2nd Edition*. 2022. URL: https://blogs.rstudio.com/tensorflow/posts/2022-05-31-deep-learning-with-R-2e/.

[Met23] Chawin Metah. "A Parallel Computing Approach for Identifying Retinitis Pigmentosa Modifiers in Drosophila Using Eye Size and Gene Expression Data". PhD thesis. Purdue University, 2023.

[Mic13] Stephen Weston Michael J. Kane John W. Emerson. "Scalable Strategies for Computing with Massive Data". In: *Journal of Statistical Software* 55.14 (2013), pp. 1–19. URL: https://www.jstatsoft.org/article/view/v055i14.

[Mor24] Mordor Intelligence Research and Advisory. *High Performance Data Analytics Market Size*. 2024. URL: https://www.mordorintelligence.com/industry-reports/high-performance-data-analytics-market (visited on 01/15/2024).

[MW11] Ethan McCallum and Stephen Weston. *Parallel R.* " O'Reilly Media, Inc.", 2011.

[Nor+22] Emily Nordmann et al. "Data Visualization Using R for Researchers Who Do Not Use R". In: *Advances in Methods and Practices in Psychological Science* 5.2 (2022), p. 25152459221074654. DOI: 10.1177/25152459221074654. eprint: https://doi.org/10.1177/25152459221074654. URL: https://doi.org/10.1177/25152459221074654.

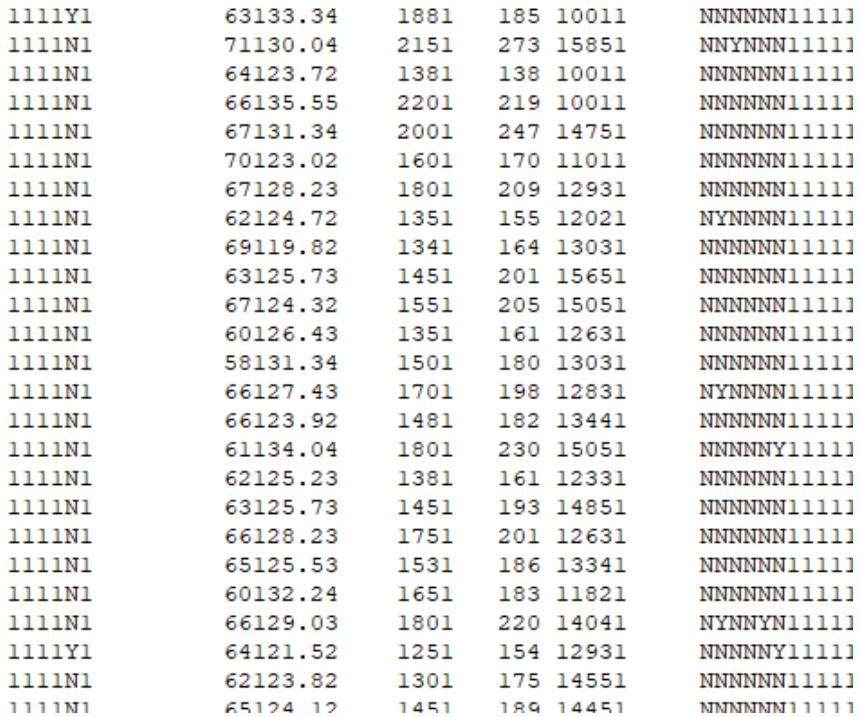
[The23] The R Foundation. *R-Project.* 2023. URL: https://www.r-project.org/about.html (visited on 01/15/2024).

[WA22] Caroline J Wendt and G. Brooke Anderson. "Ten simple rules for finding and selecting R packages". In: *PLoS Computational Biology* 18 (2022). URL: https://api.semanticscholar.org/CorpusID:247675731.

[Wic19] H. Wickham. *Advanced R, Second Edition.* Chapman & Hall/CRC the R Series. CRC Press, 2019. ISBN: 978-1-351-20130-8.

[Xu] Ruoyong Xu. "Statistical Computing With Graphics Processing Units". In: ().

A Code with Output



```
1111Y1          63133.34    1881    185 10011      NNNNNN11111
1111N1          71130.04    2151    273 15851      NNYNNN11111
1111N1          64123.72    1381    138 10011      NNNNNN11111
1111N1          66135.55    2201    219 10011      NNNNNN11111
1111N1          67131.34    2001    247 14751      NNNNNN11111
1111N1          70123.02    1601    170 11011      NNNNNN11111
1111N1          67128.23    1801    209 12931      NNNNNN11111
1111N1          62124.72    1351    155 12021      NYNNNN11111
1111N1          69119.82    1341    164 13031      NNNNNN11111
1111N1          63125.73    1451    201 15651      NNNNNN11111
1111N1          67124.32    1551    205 15051      NNNNNN11111
1111N1          60126.43    1351    161 12631      NNNNNN11111
1111N1          58131.34    1501    180 13031      NNNNNN11111
1111N1          66127.43    1701    198 12831      NYNNNN11111
1111N1          66123.92    1481    182 13441      NNNNNN11111
1111N1          61134.04    1801    230 15051      NNNNNY11111
1111N1          62125.23    1381    161 12331      NNNNNN11111
1111N1          63125.73    1451    193 14851      NNNNNN11111
1111N1          66128.23    1751    201 12631      NNNNNN11111
1111N1          65125.53    1531    186 13341      NNNNNN11111
1111N1          60132.24    1651    183 11821      NNNNNN11111
1111N1          66129.03    1801    220 14041      NYNNYN11111
1111Y1          64121.52    1251    154 12931      NNNNNY11111
1111N1          62123.82    1301    175 14551      NNNNNN11111
1111N1          65124.12    1451    189 14451      NNNNNN11111
```

Figure 4: Picture of raw data. All columns have different spacing and data types; no header or separator exists. About 50 columns and millions of rows. It is a 5 GB text file[Dri].

```r
library(readr) # import data
library(dplyr) # transform data
data <- read_fwf("Nat2018PublicUS.c20190509.r20190717.txt",
                 # specify column names and locations in the source
                 col_positions = fwf_cols(Month = c(13,14),
                 Ane = c(537,537), Men = c(538,538),
                 Cya = c(539,539), Her = c(540,540),
                 Omp = c(541,541), Gas = c(542,542),
                 Lim = c(549,549), Cle = c(550,550),
                 Pal = c(551,551), Dow = c(552,552),
                 Chr = c(553,553), Hyp = c(554,554)),
                 # specify the data type
                 col_types = "iffffffffffff") %>%
  # group by month
  group_by(Month) %>%
  # count occurrences
  summarise(Ane = sum(Ane == "Y"), Men = sum(Men == "Y"),
            Cya = sum(Cya == "Y"), Her = sum(Her == "Y"),
            Omp = sum(Omp == "Y"), Gas = sum(Gas == "Y"),
            Lim = sum(Lim == "Y"), Cle = sum(Cle == "Y"),
            Pal = sum(Pal == "Y"), Dow = sum(Dow == "P"),
            Chr = sum(Chr == "P"), Hyp = sum(Hyp == "Y"))
```

Listing 4: Reading data from figure 4 with `readr` and manipulating it with `dplyr`. Note that data frames are already part of basic R. The functionalities of Python `pandas` and `numpy` are included.

Table 3: Dataframe after importing file from figure 4 with code from listing 4.

Month	Ane	Men	Cya	Her	Omp	Gas	Lim	Cle	Pal	Dow	Chr	Hyp
1	29	55	172	46	39	73	48	183	77	103	102	174
2	25	45	175	35	31	55	34	142	81	115	100	180
3	31	48	182	41	47	72	40	200	86	90	96	180
4	34	45	186	36	32	75	42	173	56	87	90	193
5	33	40	187	46	24	80	35	180	75	91	100	197
6	34	48	189	35	33	75	45	154	74	102	100	182
7	26	43	198	34	21	74	36	179	79	86	92	193
8	24	41	189	44	43	62	48	183	88	109	94	194
9	34	44	147	40	37	66	36	158	73	112	103	196
10	25	43	207	45	31	65	49	181	77	108	115	220
11	36	55	188	39	39	62	43	144	68	98	79	173
12	23	48	196	31	31	71	31	177	86	86	73	156

```r
library(tidyr)
data_vis <- data %>%
  # make the table longer to visualize every birth anomaly
  pivot_longer(cols = 2:13, names_to="anomalies", values_to = "cases")

# plotting
library(ggplot2)
ggplot(data_vis, aes(y = cases, color = anomalies)) +
  geom_boxplot()
```

Listing 5: Rearranging data with `tidyr` and creating a simple visualization with `ggplot2`.

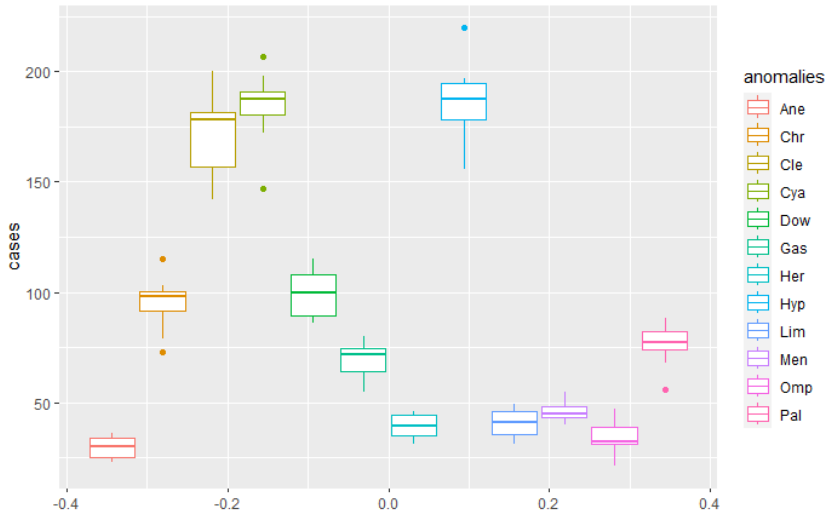


Figure 5: Visualization created with code from listing 5.

```r
data_vis %>%
  filter(anomalies %in% c("Dow", "Chr")) %>%
ggplot(aes(x = Month, y = cases, color = anomalies)) +
  geom_point() +
  geom_smooth(method = "lm", se=FALSE) +
  coord_polar() +
  facet_grid(.~anomalies) +
  scale_x_continuous("Month", breaks = c(1,2,3,4,5,6,7,8,9,10,11,12)) +
  theme_minimal()
```

Listing 6: Creating a more complex visualization with `ggplot2`.

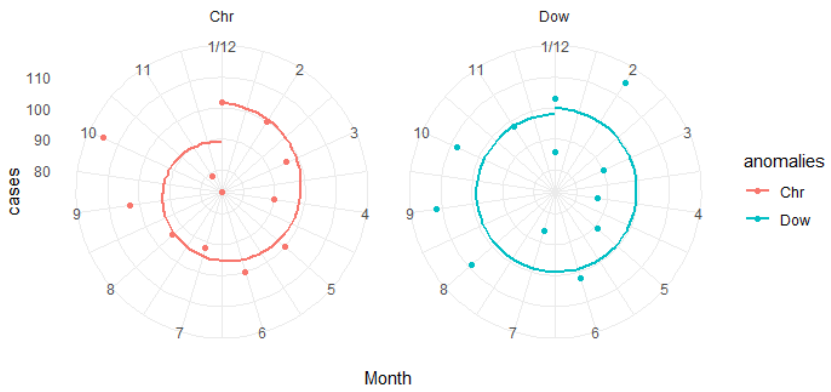


Figure 6: Visualization created with code from listing 6.

```r
# Custom R sum
sumR <- function(x) {
  total <- 0
  for (i in seq_along(x)) {
    total <- total + x[i]
  }
  total
}

# C sum
cppFunction('double sumC(NumericVector x) {
  int n = x.size();
  double total = 0;
  for(int i = 0; i < n; ++i) {
    total += x[i];
  }
  return total;
}')

# benchmark
x <- runif(1e3)
bench::mark(
  sum(x),
  sumC(x),
  sumR(x)
)[1:6]
#> # A tibble: 3 x 6
#>   expression      min   median `itr/sec` mem_alloc `gc/sec`
#>   <bch:expr> <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
#> 1 sum(x)       2.51µs   2.81µs   336848.        0B      0
#> 2 sumC(x)      3.59µs   5.24µs   177706.    2.49KB     17.8
#> 3 sumR(x)     27.14µs  29.11µs    32752.  182.59KB      0
```

Listing 7: Rewrting a loop and benchmarking it[Wic19]. The rewritten C++code is much faster than the custom R implementation.

B AI Usage Card

AI Usage Card for *Using R for High-Performance Data Analytics*

CORRESPONDENCE
Christopher L. Lübbers

CONTACT(S)
c.luebbers@stud.uni-goettingen.de

AFFILIATION(S)
Institute of Computer Science

PROJECT NAME
Using R for High-Performance Data Analytics

KEY APPLICATION(S)
The tasks and applications the project.

MODEL(S)
ChatGPT
Grammarly

DATE(S) USED
2024-03-20
2024-03-20

VERSION(S)
GPT-4
2024-03-20

IDEATION
ChatGPT

IMPROVING EXISTING IDEAS
I let ChatGPT improve my outline for the presentation. A similar outline was used in this report.

LITERATURE REVIEW
ChatGPT

FINDING LITERATURE
I searched for literature supporting my section topics.

ADDING ADDITIONAL LITERATURE FOR EXISTING STATEMENTS AND FACTS
I read something on websites and let it search for scientific literature investigating these statements.

WRITING
ChatGPT
Grammarly

GENERATING NEW TEXT BASED ON INSTRUCTIONS
Bullet points for conclusions for sections were generated by ChatGPT.

ASSISTING IN IMPROVING OWN CONTENT
My text was improved for correctness, clarity, engagement, and delivery by Grammarly.

PARAPHRASING RELATED WORK
Found literature was summarized in bullet points.

ETHICS

WHAT ARE THE IMPLICATIONS OF USING AI FOR THIS PROJECT?
Literature search was immensely simplified. The language was optimized to make it suitable in an academic context.

WHAT STEPS ARE WE TAKING TO MITIGATE ERRORS OF AI FOR THIS PROJECT?
Content was checked for correctness.

AI Usage Card v1.0 `https://ai-cards.org` PDF | BibTeX | XML | JSON | CSV

Verification of examination registration in FlexNow

Name: Mr Christopher Lee Lübbers
Matriculation No.: 20676955

Semester: WS23/24
Degree Course: Angewandte Data Science (Master of Science)
Module: M.Inf.1237: Seminar Neueste Trends in High-Performance Data Analytics
Exam: M.Inf.1237.Mp: Seminar Newest Trends in High-Performance Data Analytics (Presentation (approx. 35 min.) and report (max. 15 pages))
LV-Titel: Seminar: Newest Trends in High-Performance Data Analytics
Exam Date: 25.03.2024
Lecturer: Prof. Dr. Julian Kunkel

Declaration

I hereby declare that I have produced this work independently and without outside assistance, and have used only the sources and tools stated.

I have clearly identified the sources of any sections from other works that I have quoted or given in essence.

I have complied with the guidelines on good academic practice at the University of Göttingen.

If a digital version has been submitted, it is identical to the written one.

I am aware that failure to comply with these principles will result in the examination being graded “nicht bestanden”, i.e. failed.

Göttingen, 4th March 2024 Christopher Lee Lübbers