Technical Solutions of Developing Advanced Inftheo New Module for R

Narek Pahlevanyan
Gyumri State Pedagogical Institute
Gyumri, Armenia
e-mail: narek@ravcap.com

Mariam Haroutunian
Inst. for Informatics and Automation Problems
National Academy of Sciences of Armenia
Yerevan, Armenia
e-mail: armar@ipia.sci.am

ABSTRACT
The large volume of distributions in complex formulas of Information Theory makes computations very difficult in practice. To perform computations of complex information-theoretical results of Information Theory the authors have developed a new module (Advanced Inftheo) for R. Currently R has some associated issues, such as performance limitations, memory management restrictions, etc. In this paper we present technical solutions that are used inside Advanced Inftheo to overcome those and other limitations that have occurred during the development of the module.

Keywords
R language, R package, rate-reliability, shootout benchmarks, multithreading.

1. INTRODUCTION
Nowadays R language is evolving very fast. Large companies are currently using R for various applications. Facebook uses R for exploratory data analysis, experimental analysis, big-data visualization, human resources and user behaviour analysis related to status updates and profile pictures.
Recent interest in the financial sector has stimulated Oracle to support R, for instance, Oracle is now bundling it as part of its Big Data Appliance product. Google uses R for advertising effectiveness, economic forecasting, and big-data statistical modeling. Twitter uses R for data visualization and semantic clustering. R is a language for statistical computing, data manipulation, data mining and graphics. Robert Gentleman and Ross Ihaka started the development of R in 1993, but only recently it became popular, particularly for data scientists, as it contains a number of built-in functions for organizing data, running calculations on the information and creating elegant graphical representations of data sets. R provides a lot of different techniques for statistical linear and nonlinear modeling, time-series analysis, classification, clustering as well as graphical packages for creating high quality, and sophisticated, customized plots with very simple syntax. The capabilities of R can be extended through user-created modules. Modules are libraries developed in C++, that include specific functions for certain applications. A core set of packages included with the installation of R, with more than 5,800 additional packages and 120,000 functions are free available for download [1].
Authors believe that R can be very handy and helpful for calculations of complex formulas of Information Theory. Many information-theoretical results are difficult for computing in practice because of the large volume of distributions. For example, the investigation of rate-reliability function [2] for various applications [3], [4] is complex and computational results are complicated to obtain. Because of the difficulties in computations of rate-reliability function, the problem was solved only for particular cases, such as for simple Discrete Memoryless Channel the analytic form of the function is unknown, only the upper and lower bounds are known. R already had an extension for calculating various measures of Information Theory, but there was a need in creation of a new module for estimation and computation of more complicated formulas mentioned above.
To perform computations of complex information-theoretical results in Information Theory the authors have developed a new module for R, called Advanced Inftheo. Module Advanced Inftheo was developed in C++; it provides functionality for computation of the lower and upper bounds of rate-reliability function, as well as functionality for computation of mutual information, conditional mutual information, Kullback - Leibler (divergence) distance and other quantities of Information Theory. Furthermore, the authors have developed an option for module to connect with cluster (using the library Open MPI [5]) and execute all computational functions on cluster. This option can speed up the computation process multiple times.
The Advanced Inftheo module experimentation results are published in [6]. Specifically, in [6] the authors have computed the lower and upper bounds of E-achievable secret key rate of the biometric generated secret key sharing system for various distributions. Moreover, they provide graphical representations of the experimentation results to simplify the solutions in building of applications.
Unfortunately, R has some associated issues, such as performance limitations, memory management restrictions, etc. Those issues affected Advanced Inftheo module as well. In this paper we present technical solutions that were used to overcome limitations of R.

2. R LIMITATIONS
2.1. Performance
R is not a fast language. This is not an accident. R was designed specifically to make data analysis and statistics easier [7], [8]. It was not designed to make "life" easier for computer. In R, function arguments are evaluated only if they are actually used. To implement such evaluation, R uses a specific object that contains the expression needed to compute the result and the environment in which to perform the computation. Creating these objects has some overhead, so each additional argument to a function decreases its speed a little. Furthermore, R is a dynamic programming language and almost anything can be modified after it is created. For example, the user can change the body, arguments, and the environment of functions, modify objects outside the local environment, etc. The disadvantage of dynamism is that it is hard to predict exactly what will happen with a given function call. This is a problem because the easier it is to

predict what is going to happen, the easier it is for an interpreter or compiler to make an optimization. If an interpreter can't predict what is going to happen, it has to consider many cases before it finds the right one. The time consumption of finding the right method is higher for non-primitive functions.

To compare performance of R with C++ and Python we used the Shootout benchmarks. Results appear in Fig. 1.

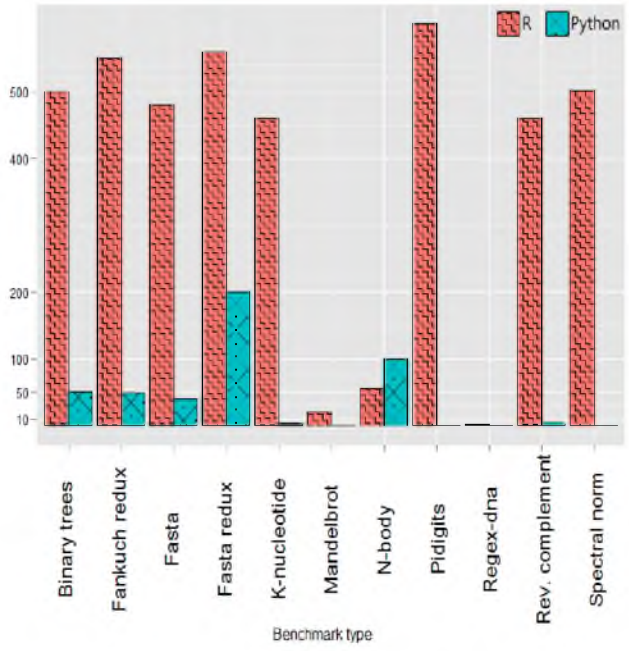

Fig. 1. Slowdown of Python and R, normalized to C++ for the Shootout benchmarks.

On those benchmarks, R is on average 501 times slower than C++ and 43 times slower than Python. Benchmarks where R results are better, like *regex-dna* (only 1.6 slower than C++), are usually cases where R transfers most of its work to C++ functions. Another big restriction that affects the performance is the absence of multithreading support inside R.

2.2. Memory management

R has memory management limitations. R memory limits depend mainly on the build of OS, but for a 32-bit build of R on Windows they depend on the underlying OS version. R holds all objects in virtual memory, and there are limits based on the amount of memory that can be used by all objects; there may be limits on the size of the heap and the number of cons cells allowed. There is also a limit on the (user) address space of a single process such as the R executable. The environment may impose limitations on the resources available to a single process: Windows versions of R do so directly. Error messages beginning "*cannot allocate vector of size*" indicate a failure to obtain memory, or because the size exceeded the address-space limit for a process. Note that on a 32-bit build there may well be enough free memory available, but not a large enough contiguous block of address space into which to map it. There are limits on individual objects. The storage space cannot exceed the address limit, and if you try to exceed that limit, the error message begins with the text "*cannot allocate vector of length*".

Furthermore, R consumes significant amounts of memory. Unlike C++, where data can be stack allocated, all user data in R must be heap allocated and garbage collected. Comparison of heap memory usage in C++ (calls to *new/malloc*) and data allocated by the R virtual machine is in Fig. 2.

The figure shows that R allocates a lot more data than C++, and is clearly memory inefficient.

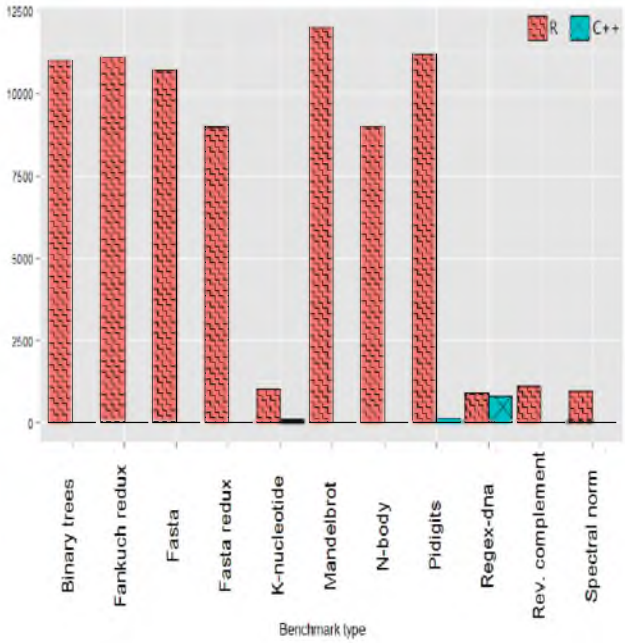

Fig. 2. Heap allocated memory (MB log scale). C++ vs. R.

3. TECHNICAL SOLUTIONS

3.1. Sequential and multithread functions

It was mentioned above that Advanced Inftheo module was developed in C++; it suggests that R limitations can be solved using different techniques available for C++. Fig. 3 illustrates Advanced Inftheo main functions slowdown compared with versions of the same functions that were running under pure C++. Note that the figure depicts function versions that are executed in both environments only using a single thread.

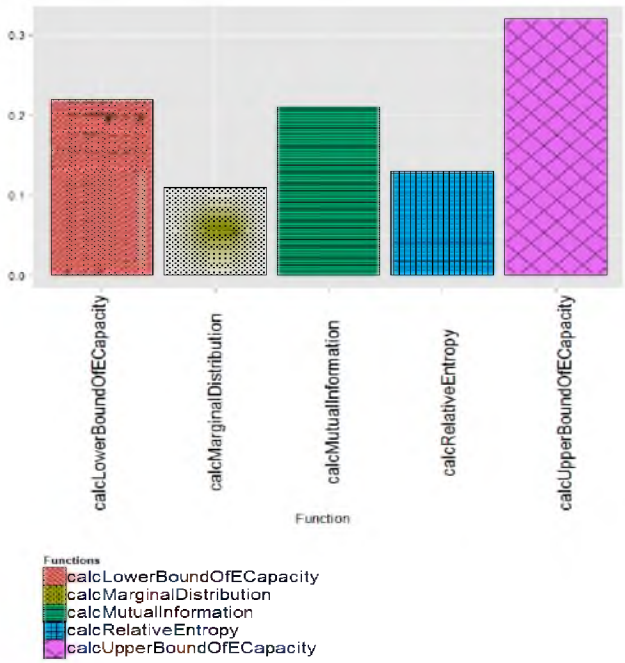

Fig. 3. Slowdown of Advanced Inftheo main functions from R environment, normalized to C++ versions of the same functions.

From Fig. 3. we can see that performance of functions in Advanced Inftheo is very close to the performance of the same functions that have been executed in pure C++. The slowdown comes mostly because some functions of Advanced Inftheo are hybrids and are using R functions too. Moreover, Advanced Inftheo module is using multithreading for faster performance [9]. That means that Advanced Inftheo can take advantage of multiprocessor hardware.

Usage of multithreading also implies that new issues arise with execution of parallel threads. At first sight multithreaded programming seems rather simple. Instead of having just one processing unit for performing the work, you have two or more executing simultaneously. Because the processors might be real hardware, the term "thread" is used instead of the processor.

A thread is a path of execution through the program. In a single threaded program, there is always a single path of execution. While in a multithreaded program, there are two or more paths of execution. It means, that while using a single threaded program, only one task can be executed at a time, and the program waits until the task is finished/completed, before starting another one. For most uses, one thread of execution is all that is needed, but for our case, gain in execution time is very crucial. For specific tasks, the ability to use multiple threads in parallel can result in significant performance gains. The tricky part of multithreaded programming is how the threads communicate with each other.

The most commonly deployed multithreaded communication model is called a shared memory model [10]. In this model all threads have an access to the same pool of shared memory. The advantage of this model is that multithreaded programs are programmed in much the same way as sequential programs like R. That advantage, however, is its biggest problem. The model does not distinguish between memory that is being used strictly for thread local use (like most locally declared variables), and memory that is being used to communicate with other threads (like some global variables and heap memory). Since the memory that is potentially shared, needs to be treated much more carefully than the memory that is local to a thread, it's becoming much easier for making mistakes. The most common way of preventing access to the same shared resources is to use locks to prevent the other threads from accessing memory associated with an invariant. Locks are one of the key techniques that are being used inside Advanced Inftheo. Lock gains different names. It is sometimes called a monitor, a critical section, a mutex, or a binary semaphore, but regardless of the name, it provides the same basic functionality. The lock provides enter and exit points, and once a thread calls enter, all attempts by other threads to call enter will cause the other threads to wait until a call to exit is made. The thread called enter is the owner of the lock, and it is considered a programming error if exit is called by a thread that is not the owner of the lock. Locks provide a mechanism for ensuring that only one thread can execute a particular region of code at any given time.

Memory can be made safe for multithreaded use in several ways. Firstly, memory that is only accessed by a single thread is safe because other threads are unaffected by it. This includes most local variables and all heap-allocated memory before it is published.

Secondly, memory that is read-only after publication does not need a lock because any invariants associated with it must hold for the rest of the program.

Thirdly, memory that is actively updated from multiple threads generally uses locks to ensure that only one thread has an access while a program invariant is broken.

Finally, in certain specialized cases where the program invariant is relatively weak, it is possible to perform updates that can be done without locks. In Advanced Inftheo specialized compare-and-exchange instructions are used. These techniques are one thought of as special implementations of locks.

3.2. Deadlocks

One of the big reasons for avoiding many locks in the module is deadlock [9]. Once a module has more than one lock, deadlock becomes a possibility. For example, if one thread tries to enter Lock X and then Lock Z, while at the same time another thread tries to enter Lock Z and then Lock X, it is possible for them to deadlock if each enters the lock that the other owns before attempting to enter the second lock.

From a pragmatic perspective, deadlocks are generally prevented in one of the two ways. The first way to prevent deadlock, is to have enough locks in the system that it is never necessary to take more than one lock at a time. If this is impossible, deadlock can be prevented by having a convention on the order in which locks are taken. Deadlocks can only occur if there is a circular chain of threads such that each thread in the chain is waiting on a lock already acquired by the next in line. To prevent this in Advanced Inftheo each lock in the system is assigned a "level", and the functions are designed so that threads always take locks only in strictly descending order by level. This protocol makes loops involving locks, and therefore deadlock is impossible. Deadlocks are just another reason for keeping the number of locks in the system small.

3.3. Synchronization

While locks provide a way of keeping threads out of each other's way, they don't provide a technique for them to synchronize. Generally, events are used as a signal that a more complicated function property holds. For example, a function might have a queue of work for a thread, and an event is used to signal to the thread that the queue is not empty. The rules for proper locking require that if code needs a resource, there must be locks that provide exclusive access for all memory associated with that resource. Applying this principle in a queue suggests that only after entering a common lock all the accesses to the event and the queue should happen.

Unfortunately, this design can cause a deadlock. For example. Thread X enters the lock and needs to wait for the queue to be filled. Thread Z, which is attempting to add an entry to the queue that Thread X needs, will try to enter the queue's lock before modifying the queue and thus block on Thread X. A common practice is to release the lock and then wait on the event. A typical solution was to weaken the resource in this case to, "if the event is reset, then the queue is empty." This resource is strong enough that it is still safe to wait on the event without risking waiting forever. The waking thread has to enter the queue's lock and verify that the queue has an element. If say some other thread removed the entry, it must wait again. If fairness among the threads is important, this solution has a problem, but it does work well for all functions inside Advanced Inftheo.

3.4. Implementation on cluster (Open MPI)

As already mentioned for speeding up computational process we have built an option for execution of module functions on cluster. The key point of implementation of this feature was attachment of Open MPI library to the module. The main reason why we chose Open MPI is it's precisely designed

Modular Component Architecture (MCA), which breaks all of the Advanced Inftheo functionality into narrowly grouped modules that can be modified independently [11]. The MCA administrates the component frameworks and provides services to them, such as the ability to accept run-time parameters from higher-level abstractions and pass them down through the component framework to individual components. Each component framework is devoted to a particular task, such as providing parallel job control or perform MPI collective operations. Framework also can discover, load, use, and unload components. Each framework has different usage scenarios; some will only use one component at a time, while the others will use all the available components at the same time. Components are software units that can configure, build, and install themselves. Open MPI uses a flexible component architecture, and it's point-to-point design is such that it provides excellent communication performance for different interconnections inside Advanced Inftheo functions.

4. CONCLUSION

Thus, the technical approaches used in Advanced Inftheo module allowed to achieve high performance and dynamic memory management.
Parallelism inside Advanced Inftheo can give huge time gain in computations of information-theoretical results for the practical applications.

REFERENCES

[1] W. N. Venables, D. M. Smith and the R Core Team, "An Introduction to R", version 3.1.1, pp. 51-77, 2014.

[2] E. A. Haroutunian, M. E. Haroutunian, and A. N.Harutyunyan, "Reliability criteria in information theory and in statistical hypothesis testing, " Foundations and Trends in Communications and Information Theory, Vol. 4, no. 23, pp. 97-263, 2008.

[3] M. E. Haroutunian, N. S. Pahlevanyan "Information theoretical analysis of biometric secret key sharing model," Transactions of IIAP of NAS of RA, Mathematical Problems of Computer Science, vol.42, pp. 17-27, 2014.

[4] E. Haroutunian, "On bounds for *E*-capacity of DMC," IEEE Transactions on Information Theory, vol. 53, no. 11, pp. 4210–4220, 2007.

[5] Michael J. Quinn, Parallel Programming in C with MPI and OpenMP, McGraw-Hill Education Group, 2003.

[6] M. E. Haroutunian, N. S. Pahlevanyan "Experimentation of Advanced Inftheo module for R on the example of biometric generated secret key sharing system," International Journal "Information Content and Processing", Vol. 2, Number 1, pp. 62-70, 2015.

[7] R Development Core Team. "R: A Language and Environment for Statistical Computing, " R Foundation for Statistical Computing, 2011.

[8] D. Smith. "The R ecosystem". In The R User Conference 2011, August 2011.

[9] C. Hughes, T. Hughes, Professional Multicore Programming: Design and Implementation for C++ Developers, Wrox Press Ltd., Birmingham, UK, 2008.

[10] A. Polukhin, Boost C++ Application Development Cookbook, Packt Publishing, 2013.

[11] E. Gabriel, G. E. Fagg, and G. Bosilca, "Open MPI: Goals, concept, and design of a next generation MPI implementation, " in Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 2004, pp. 97–104.