D1-1 Optimisation of R bootstrapping on HECToR with SPRINT

Project Title	ESPRC dCSE "Bootstrapping and support vector machines with R and SPRINT"
Document Title	D1-1 Optimisation of R bootstrapping on HECToR with SPRINT
Authorship	Michal Piotrowski
Document Filename	SPRINT-dCSE3-boostrap-D1-1_report_v0.2
Document Version	0.2
Document Number	
Distribution Classification	Public following PI permission

Document History

Personnel	Date	Comment	Version
T. Sloan	14/01/2013	Renamed file and added frontpage for submission to NAG	0.2
M.Piotrowski	08/05/2012	Original draft	0.1

Contents

Introduction

They key objective of this dCSE project was to implement a parallel version of the bootstrapping method, a very popular and computationally demanding resampling method used to measure accuracy of a sample estimates and assisting with statistical inference. A user requirements survey conducted at the beginning of the SPRINT project, highlighted the bootstrap function as one of the top candidates for parallelisation.

The Simple Parallel R Interface (SPRINT) [1], the project of the Division of Pathway Medicine and EPCC at The University of Edinburgh, allows R users to exploit the HECToR supercomputing facility and other High Performance Computing (HPC) systems without expert knowledge of such systems. R is a freely available language and environment for statistical computing (http://www.r-project.org/).

As the result of this project the new 1.0 version of the SPRINT framework was released on The Comprehensive R Archive Network (CRAN). This required some updates of the existing code to be fully compliant with the R 2.15.0 coding standards.

The Bootstrap

The bootstrapping method, introduced by Efron in late 70's, is strictly linked with the development of computing systems capabilities. It is a general technique used to determine uncertainty in population estimation, replacing mathematical analysis with computer simulation. This is achieved by resampling with replacement the original observation data and thus generating new samples of equal length but highlighting different distribution properties of the original data. The higher number of resamples improves accuracy of the final estimation, but requires a lot of processing power. In the era of data intensive research, the number of bootstrap resamples is usually limited by the available computing resources. The bootstrap technique is a popular method that has a wide range of possible applications, it is computationally intensive and fairly easy to parallelize, which makes it perfect addition to the SPRINT library.

The best way to explain how the bootstrap works is with an example. Let's assume that we would like to estimate the mean M from unknown population P on the basis of randomly sampled data. We can calculate the sample mean m, but we would also like to estimate the standard error of m to assess the amount of uncertainty in our estimate, which depends on our population P variance that is unknown. The idea behind bootstrap is that we could simulate the entire population distribution using just the sample provided. We assume that both sample and population have similar shape and generate new samples by resampling the original with replacement. This introduces variability into our measurements. We can now estimate the variance of P by calculating standard deviation of multiple m values obtained using the bootstrap technique.

The boot function that allows users to generate bootstrap samples in R is implemented in the boot package. It can bootstrap any statistical function that can be expressed in R and from these samples it can generate estimates of bias and bootstrap confidence intervals. The boot command executes the resampling of your dataset and calculation of your statistic of interest on these samples.
Below is the example of the boot call taken from the documentation document supplied with the package. The first argument passed to the function should be your dataset. In this case it is a city dataset distributed with the boot package. The second argument can be an index vector of the observations in your dataset to use or a frequency or weight vector that informs the sampling probabilities. The example below uses the vector of importance weights, that is passed as the second argument to the ratio function, and by default it will use all values in the data set. The statistic to be bootstrapped is defined as the *ratio* function and it is the 49 U.S. cities population increase ratio between 1920 and 1930. This is how we define the statistic:

```
> library(boot)
> ratio <- function(d, w) sum(d$x * w)/sum(d$u * w)
```

Now we can call the boot command, passing our data set, our own statistic function, number of replicates and the statistic type.

```
> boot(city, ratio, R = 999, stype = "w")
> quit()
ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = city, statistic = ratio, R = 999, stype = "w")


Bootstrap Statistics :
    original     bias    std. error
t1* 1.520313 0.03959516   0.2203012
```

The parallel interface is almost identical, only instead of boot function we add p in front of the function name. We also have to load sprint library instead of boot and terminate the MPI after all computation is finished.

```
> library(boot)
> ratio <- function(d, w) sum(d$x * w)/sum(d$u * w)

> pboot(city, ratio, R = 999, stype = "w")

> pterminate()
> quit()
```

Implementation

The first prototype version of the parallel bootstrap *pboot*, was implemented as part of the MSc project at EPCC. This early version was a good capability test for the old SPRINT architecture where only one process could interact with the R runtime environment and thus was not able to benefit from accessing the R interpreter simultaneously on all process involved in computation. As the result it was not compatible with the rest of the SPRINT functions and could not be combined in the parallel workflows. Furthermore, due to inefficient data distribution it only scaled up to 8 processors on the Hector (phase 3) machine. It also implemented only a fraction of the original bootstrap functionality, it was limited only to the non-parametric standard simulation.

The new version of SPRINT spawns individual R process and associated runtime environment on all nodes. This significantly simplifies the parallelisation of more complex functions written partially or purely in R. The body of the function no longer has to be rewritten in C, but can be serialised and send to all the nodes. It is then deserialised and passed back to the R interpreter that carries out the computation. The parallelisation is achieved by data distribution with each node executing the same function on a fraction of the data set. This approach can be only applied in situations where no data dependency occurs.

This model also allows the sending of more complex R objects between the nodes, as well as user defined functions that are passed as arguments. This approach ideally suits our parallel bootstrap implementation, where a copy of a given function is executed on each process using a number of different samples of the data. There is no data dependency between each function run and the bootstrapped statistics is not known in advance, thus cannot be rewritten in C. Furthermore, this approach will allow us to keep the identical interface as in the serial version, we will serialise the original function call including all the arguments and execute it using the number of resamples on all the participating nodes.

There is a number of possible simulation methods that can be generated using the sequential *boot* function. These are "ordinary", "balanced", "antithetic", "permutation", "parametric". Various auxiliary functions find the indices to be used for the bootstrap replicates and then statistic function loops over those replicates. The process of generating new indices is currently performed in the serial part of the code and then the new replicates are redistributed to all the workers. This definitely has impact on the performance and scalability, but was chosen for simplicity and to allow the complete reproducibility of results when compared with the sequential version.

At the end of each iteration a single bootstrap statistic is calculated on each process and added to a local SEXP list. After all computation is finished these lists are combined using the SPRINT parallel reduce function, initially implemented to speedup tree reduction algorithm in SPRINT's parallel random forest implementation. The general implementation allows to provide our own combine function that will be passed as an argument to the parallel reduce function as long as it is associative. This way our parallel combine algorithm will concatenate result lists with logarithmic rather than linear complexity, improving the performance significantly.

The interface of the parallel reduction function is as follows:

```
void reduce(SEXP in, SEXP *out, SEXP (*combine_fn)(SEXP, SEXP), int root,
MPI_Comm comm)
```

The parallel bootstrap interface is identical with its sequential counterpart. The minimal changes required to run parallel version include loading the sprint library, renaming *boot* function call to *pboot* and terminating MPI environment at the end of the script.

Results

The performance results were gathered on the current Phase 3 system (XE), 2816 computing nodes, each contains two 16-core AMD Opteron 2.3Ghz Interlagos processors, which gives a total number of 90,112 available cores. All benchmark runs were performed using data from Golub et al. These are the combined training samples and test samples. There are 47 patients with acute lymphoblastic leukemia (ALL) and 25 patients with acute myeloid leukemia (AML), data on the expression of 7129 genes are available. We have performed bootstrap using three different statistical functions, trying to determine how scalability depends on function complexity and number of replicates.

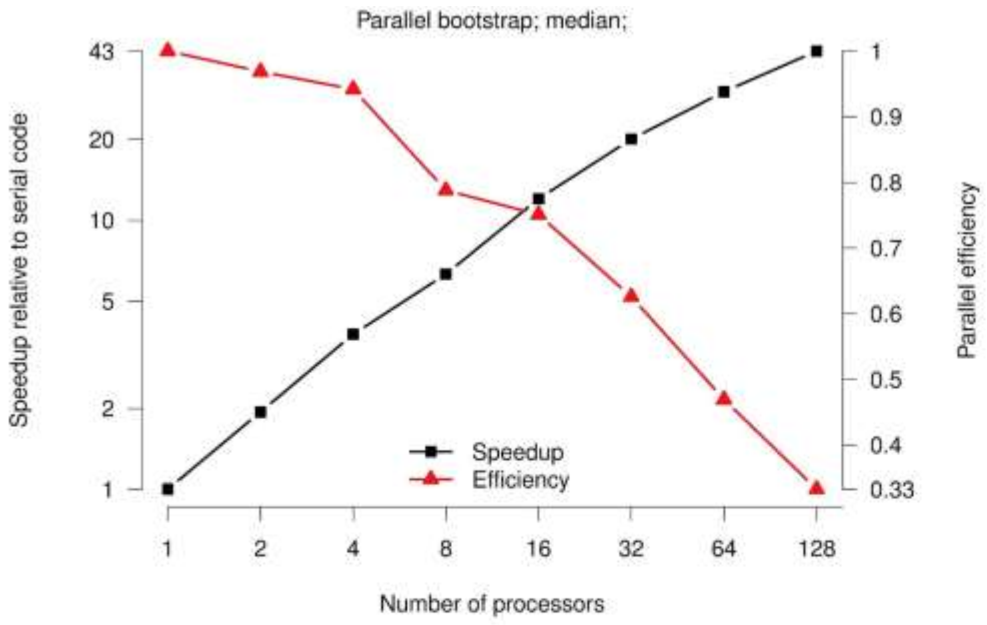

Figure 1 results of the sample median, using the full Golub_Merge dataset and 24999 replicates.

Both sample median and standard deviation are functions of fairly low complexity. The main difference between these two runs is the number of replicates used for bootstrapping. The both benchmarks show fairly mediocre performance with parallel efficiency lowering after 16 processors for greater number of resamples in Figure 1 and after 8 where only 9999 replicates are used to calculate standard deviation estimates. The decision to generate random indices on the Master process was unfortunate in this case, as the short amount of time spent on calculating simple statistics in parallel is being overshadowed by the time spent in the sequential part. The impact of calculating and distributing the indices is especially visible in Figure 1, where we can observe steep drop in performance as we go beyond 4 and 16 process where the memory access becomes more remote.

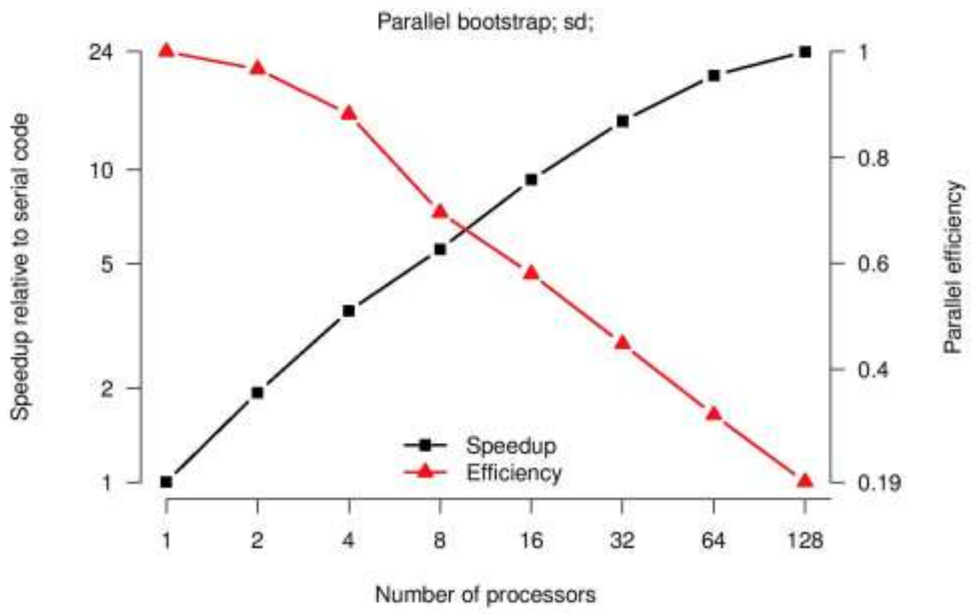

Figure 2 results of the sample standard deviation, using the full Golub_Merge dataset and 9999 replicates.

The last benchmark shows bootstrapping of the partitioning around the medoids function. We have reduced the sample size to 2500 genes and the number of replicated to 999, clustering the data set around 6 medoids. The complexity of pam function is much higher than the previous ones and we can observe more stable scalability dropping off for more than 32 processors.

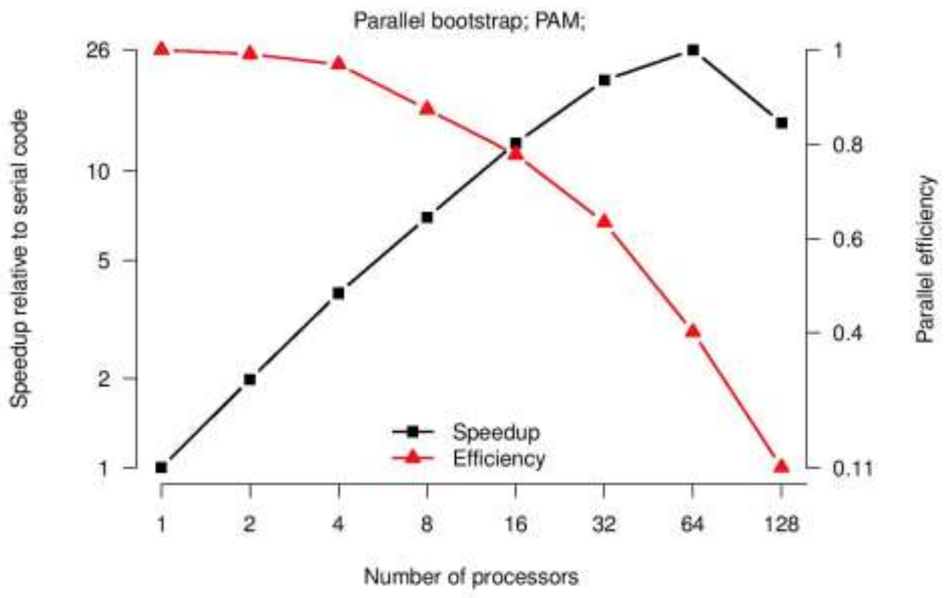

Figure 3 results of the partitioning around medoids of 2500 genes from the Golub_Merge dataset and 999 replicates.

CRAN release

As part of this dCSE project we have prepared a first full release of the SPRINT package on The Comprehensive R Archive Network called CRAN, a web repository, composed of a network of mirror sites around the world with up to date versions of the R code and the vast number of its contributed packages.
The CRAN is maintained by the core R developers group and before any new package can be submitted it must pass R check without warnings or significant notes to be admitted to the main CRAN package area. Many aspects of the check procedure have changed since R 2.15.0 release and the following modifications to the SPRINT code were required.
As of this new version the submission test doesn’t allow the compiled code to call abort or exit functions and to terminate the user’s R process in uncontrolled manner. The reason being that user might lose all his unsaved work. One usage that could call abort is the assert macro in C or C++ functions, which should never be active in production code. Unfortunately, this was not the case with some third party code that was incorporated to SPRINT when pmt.maxT function was implemented. Assertion methods were used in the production code and were part of the algorithm logic, rather a separate debug code. This was fixed for the release. However, one exit call still remains in the SPRINT implementation. This is during the SPRINT shutdown operation when worker nodes receive termination command code from the master process, after MPI is finalised all workers call exit function. Otherwise, worker nodes would return to R and continue to execute the script just after SPRINT library was loaded, calling parallel functions after MPI environment was terminated crashing the system. This problem was explained to the CRAN maintainers and they decided to make this one exception for the SPRINT code.
The new R standard doesn’t allow to write to standard output or standard error streams instead of console. The new check detects these symbols and blocks from admission any code that contains reference to these streams. The problem with this test is that the detected symbols are linked into the code but might come from some third party libraries libraries and not actually be called. We had to go through all our code and all third party libraries and remove any reference other than to R console which has fixed the problem.
Other modifications required for the CRAN release include resolving unstated dependencies in the R code, adding example calls that are run during the submission test, fixing any mismatches between the .Rd documentation files and the actual implementation and completing missing documentation entries.

Conclusions

We have presented how a parallel bootstrap version was implemented for the SPRINT package. Depending on the complexity of the statistic(s) and the number of replicates used for bootstrapping we achieved speedup between 20 and 40 compared to the original serial code on HECToR. This could be further improved by wrapping different methods of random indices generation methods and calling them in parallel from individual workers. Our current implementation has identical interface and returns result in identical R structure as the sequential boot version, making it very easy to adapt for existing R scripts. Furthermore, we have performed some code refactoring, make it compatible with the R 2.15.0 version, improved the user documentation and deployment procedure before releasing the new version of SPRINT on CRAN.

Acknowledgements

The SPRINT project is funded by a Wellcome Trust grant [086696/Z/08/Z]. This project was funded under the HECToR distributed Computational Science and Engineering (CSE) Service operated by NAG Ltd. HECToR – A Research Councils UK High End Computing Service – is the UK's national supercomputing service, managed by EPSRC on behalf of the participating Research Councils. Its mission is to support capability science and engineering in UK academia. The HECToR supercomputers are managed by UoE HPCx Ltd and the CSE

References

[1] J. Hill, M. Hambley, T. Forster, M. Mewissen, T. Sloan, F. Scharinger, A. Trew, and P. Ghazal. SPRINT: A new parallel framework for R. BMC Bioinformatics, 9(1):558, 2008.