parallelDist

Alexander Eckert

parallelDist version 0.2.2 as of September 24, 2018

Abstract

This document highlights the performance gains for calculating distance matrices with the **parallelDist** package and provides basic usage examples.

Contents

1 Introduction

The **parallelDist** package provides a fast parallelized alternative to R's native `dist` function to calculate distance matrices for continuous, binary, and multi-dimensional input matrices and offers a broad variety of predefined distance functions from the **stats**, **proxy** and **dtw** R packages, as well as support for user-defined distance functions written in C++. For ease of use, the `parDist` function extends the signature of the `dist` function and uses the same parameter naming conventions as distance methods of existing R packages.

The package is mainly implemented in C++ and leverages the **Rcpp** [EF11] and **RcppParallel** [AFU+16] package to parallelize the distance computations with the help of the TinyThread library. Furthermore, the Armadillo linear algebra library [San10] is used via **RcppArmadillo** [ES14] for optimized matrix operations for distance calculations. The curiously recurring template pattern (CRTP) technique is applied to avoid virtual functions, which improves the Dynamic Time Warping calculations while keeping the implementation flexible enough to support different step patterns and normalization methods.

2 Performance

The inital motivation for building this package was the need for a fast Dynamic Time Warping implementation which uses multiple cores and supports multi-dimensional (time) series. DTW

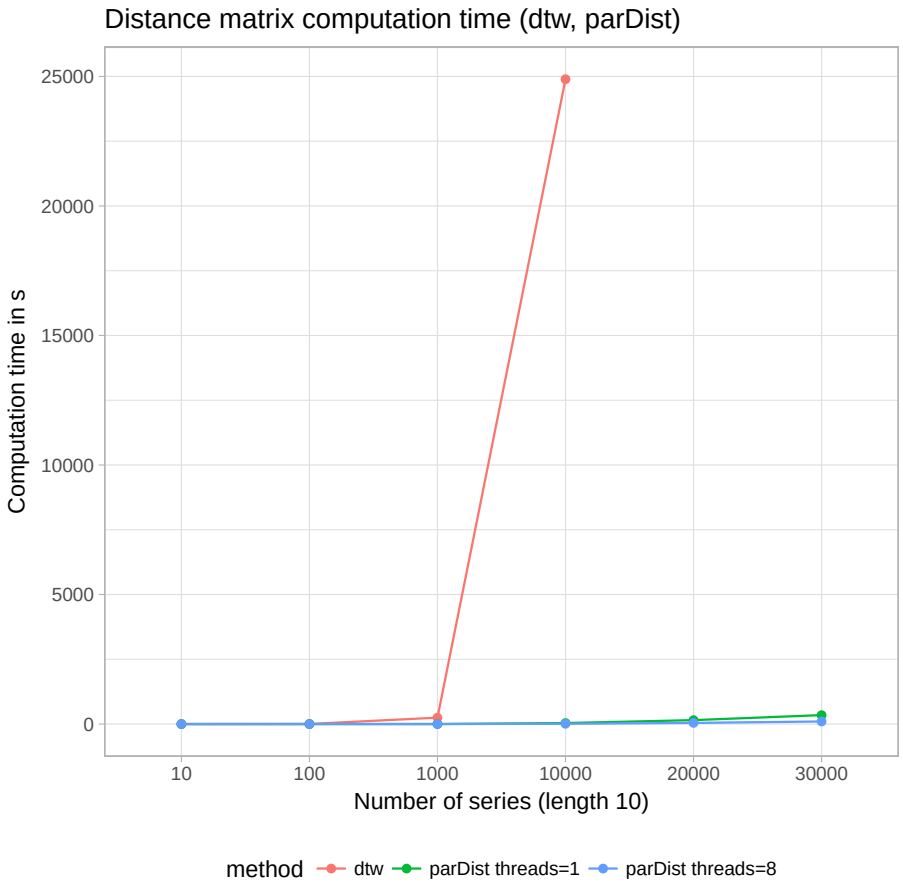

Figure 1: Distance matrix computation time for Dynamic Time Warping

is an expensive distance measure, where the computation of the DTW distance between two series of length N has a complexity of $\mathcal{O}(N^2)$. This motivates an efficient and parallelized implementation in C++.

Figure 1 shows a performance comparison between the `parDist` function of **parallelDist** and the `dist` function in conjunction with the **dtw** package.

The benchmark has been performed on a system with the following specifications:

- Intel(R) Xeon(R) E3-1230 v3 @ 3.30 GHz, 4 cores with hyper-threading
- 32 Gb RAM

As depicted in figure 1, `parDist` makes the calculation of large distance matrices with DTW up to 3 orders of magnitudes faster.

The `parDist` function can be used as a replacement for the `dist` function of the **stats** package, since it supports all other distance methods of the **stats** package and most of the distances of the **proxy** package. Figure 2 shows the performance comparison of the `parDist` function with

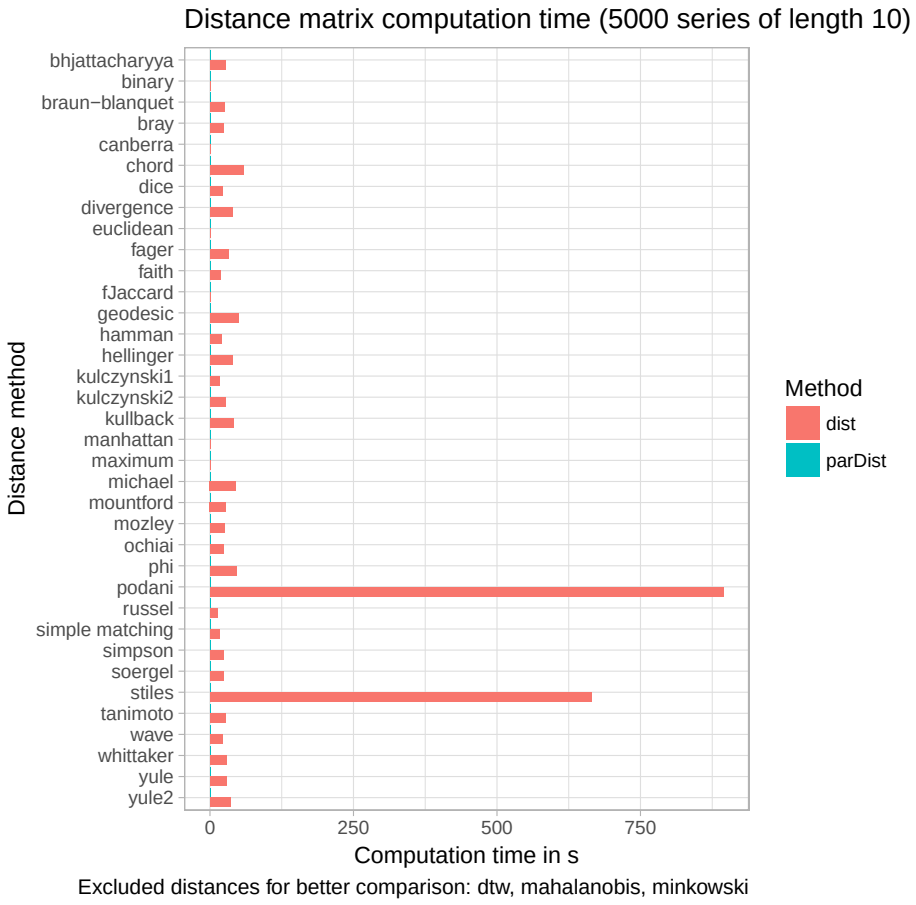

Figure 2: Distance matrix computation times

the distance methods of **stats** and the **proxy** package when calculating distance matrices with 5000 series of length 10.

3 Quick start

3.1 Using matrices as input parameter

The function signature of `parDist` is based on dist. To calculate a distance matrix for 10 series of length 10, a matrix is passed to the `parDist` function where each row corresponds to one series.

```
> # matrix where each row corresponds to one series
> sample.matrix <- matrix(c(1:100), ncol = 10)
```

Here the `parDist` function calculates the distance matrix using the euclidean distance and

returns a dist object, like the dist function.

```
> # euclidean distance
> dist.euclidean <- parDist(sample.matrix, method = "euclidean")
```

The dist object can easily converted into a matrix, or can be used as an input for R's clustering algorithms.

```
> # convert to matrix
> as.matrix(dist.euclidean)
> # create hierarchical agglomerative clustering model
> hclust.model <- hclust(dist.euclidean, method="ward")
```

Some distance methods require additional arguments (see `?parDist`). These additional arguments can be passed directly to the `parDist` function.

```
> # minkowski distance with parameter p=2
> parDist(x = sample.matrix, method = "minkowski", p=2)
> # dynamic time warping distance normalized with warping path length
> parDist(x = sample.matrix, method = "dtw", norm.method="path.length")
```

A list of all available distance methods can be found in the `parDist` documentation.

```
> ?parDist
```

The number of threads to use can be set via the threads parameter.

```
> # use 2 threads
> dist.euclidean <- parDist(sample.matrix, method = "euclidean", threads = 2)
```

3.2 Using a list of matrices as input parameter

`parDist` also supports the calculation of distances between multi-dimensional series. Instead of one single matrix a list of matrices is used as input parameter. One matrix with M rows and N columns corresponds to a series with M dimensions and length N.

In the example below, a list with 2 matrices is defined where each matrix corresponds to a series with 2 dimensions of length 10.

```
> # defining a list of matrices, where each
> # list entry row corresponds to a two dimensional series
> tmp.mat <- matrix(c(1:40), ncol = 10)
> sample.matrix.list <- list(tmp.mat[1:2,], tmp.mat[3:4,])
```

The sample matrix now can be used to calculate a distance matrix for the multi-dimensional DTW distance.

```
> # multi-dimensional dynamic time warping
> parDist(x = sample.matrix.list, method = "dtw")
```

3.3 Using user-defined distance functions

Since version 0.2.0 of **parallelDist** custom user-defined distance measures can be defined to calculate distances matrices in parallel. To ensure a performant execution, the user-defined function needs to be defined and compiled in C++ and an external pointer to the compiled C++ function needs to be passed to `parDist` with the `func` argument.

The user-defined function needs to have the following signature:

```
double customDist(const arma::mat &A, const arma::mat &B)
```

Note that the return value must be a `double` and the two parameters must be of type `const arma::mat &param`. More information about the Armadillo library can be found at [Arm] or as part of the documentation of the **RcppArmadillo** [ES14] package.

Defining and compiling the function, as well as creating an external pointer to the user-defined function can easily be achieved with the `cppXPtr` function of the **RcppXPtrUtils** package. The following code shows a full example of defining and using a user-defined euclidean distance function:

```
> # RcppArmadillo is used as dependency
> library(RcppArmadillo)
> # Use RcppXPtrUtils for simple usage of C++ external pointers
> library(RcppXPtrUtils)
> # compile user-defined function and return pointer (RcppArmadillo is used as dependency)
> euclideanFuncPtr <- cppXPtr("double customDist(const arma::mat &A, const arma::mat &B) {
+                             return sqrt(arma::accu(arma::square(A - B))); }",
+                             depends = c("RcppArmadillo"))
> # distance matrix for user-defined euclidean distance function
> # (note that method is set to "custom")
> parDist(matrix(1:16, ncol=2), method="custom", func = euclideanFuncPtr)
```

As displayed in table 1, the performance between a user-defined and a predefined distance function is close to equal for large matrices.

	matrix	method	min	lq	mean	median	uq	max	neval
1	10x10	euclidean	0.04	0.05	0.08	0.06	0.13	0.17	100.00
2	10x10	custom	0.16	0.18	0.24	0.22	0.30	0.40	100.00
3	100x10	euclidean	0.09	0.11	0.14	0.13	0.18	0.24	100.00
4	100x10	custom	0.22	0.24	0.31	0.29	0.37	0.51	100.00
5	1000x10	euclidean	4.19	4.29	4.44	4.35	4.50	5.54	100.00
6	1000x10	custom	4.34	4.48	4.66	4.60	4.80	5.14	100.00
7	10000x10	euclidean	448.87	451.33	490.99	453.26	465.36	644.65	100.00
8	10000x10	custom	452.83	454.99	492.68	456.36	464.66	678.49	100.00

Table 1: Performance comparison between user-defined and predefined euclidean distance function (in ms)

3.4 Using objects of other R packages

The `parDist` supports different kinds of step patterns for calculating DTW distance matrices (see `?parDist`). For ease of use, it is also possible to use the StepPattern objects of the **dtw** package as input parameters for `parDist`.

```
> # load dtw package
> library(dtw)
> # print the step pattern
> print(symmetric2)
> # use the symmetric2 object as input parameter for the parDist function
> parDist(x = sample.matrix, method = "dtw", step.pattern = symmetric2)
```

References

[AFU+16] JJ Allaire, Romain Francois, Kevin Ushey, Gregory Vandenbrouck, Marcus Geelnard, and Intel. *RcppParallel: Parallel Programming Tools for 'Rcpp'*, 2016. R package version 4.3.20.

[Arm] Armadillo: C++ linear algebra library documentation. http://arma.sourceforge.net/docs.html.

[EF11] Dirk Eddelbuettel and Romain François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011.

[ES14] Dirk Eddelbuettel and Conrad Sanderson. Rcpparmadillo: Accelerating r with high-performance c++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, March 2014.

[San10] Conrad Sanderson. Armadillo: An open source C++ algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, 2010.