Fast computation of recurrences in long time series

Tobias Rawald, Mike Sips, Norbert Marwan, and Doris Dransch

Abstract We present an approach to recurrence quantification analysis (RQA) that allows to process very long time series fast. To do so, it utilizes the paradigm *Divide and Recombine*. We *divide* the underlying matrix of a recurrence plot (RP) into sub matrices. The processing of the sub matrices is distributed across multiple graphics processing unit (GPU) devices. GPU devices perform RQA computations very fast since they match the problem very well. The individual results of the sub matrices are *recombined* into a global RQA solution. To address the specific challenges of subdividing the recurrence matrix, we introduce means of synchronization as well as additional data structures. Outperforming existing implementations dramatically, our GPU implementation of RQA processes time series consisting of $N \approx 1,000,000$ data points in about 5 minutes.

1 Introduction

Many different systems show recurring behavior and its study has attracted attention in almost all scientific fields. The climate system can express recurring behavior due to Milankovich cycles [1], seasonal changes, El Niño/Southern Oscillation, etc. The study of these recurrences allows for a better understanding of the climate system; its past as well as its future [2–4]. The cardiorespiratory system is investigated by its recurrence properties to get insights in its mechanisms or to measure dysfunctions for diagnosing life threatening conditions [5–9]. Recurrence analysis is promising for investigating brain activity [10] and early detection of epileptic states [11]. Furthermore, it is applied to monitor engines and technological processes, like power generation gas turbines health, cutting processes or crack detection in materials [12–14].

T. Rawald and D. Dransch, German Research Center for GeoSciences GFZ and Humboldt-Universität zu Berlin e-mail: `(trawald,dransch)@gfz-potsdam.de` · M. Sips German Research Center for GeoSciences GFZ e-mail: `sips@gfz-potsdam.de` · N. Marwan, Potsdam Institute for Climate Impact Research e-mail: `marwan@pik-potsdam.de`

Recurrence plots (RPs) and recurrence quantification analysis (RQA) are powerful methods for analyzing recurrences in measured time series [15]. Their application in many fields have proven their potential for various kinds of analyses [16]. A recurrence plot is a two-dimensional representation of a time series when a m-dimensional phase space trajectory recurs to former (or later) states. Recurrence of a state at time i at a different time j is captured within a two-dimensional squared matrix $\mathbf{r}$ [15]:

$$r_{i,j} = \Theta\left(\varepsilon - \left\|\mathbf{x}_i - \mathbf{x}_j\right\|\right), \quad \mathbf{x}_i \in \mathbb{R}^m, \quad i,j = 1\ldots N. \tag{1}$$

Both of its axes represent the set of states in temporal order. N is the number of considered states x_i (length of phase space trajectory). ε is a threshold distance, $\|\cdot\|$ a norm, and $\Theta(\cdot)$ the Heaviside function. A pair of states that fulfills the threshold condition is assigned with the value 1 (*recurrence point*), whereas a pair that is considered to be dissimilar is assigned with the value 0. Further details about the reconstruction of phase space vectors from a scalar time series, the recurrence parameters, as well as the typical visual characteristics of RPs can be found in [15].

Small scale structures in the RP, like diagonal lines, are used to define measures of complexity establishing the recurrence quantification analysis (RQA) [15,17,18]. As an example, we present the RQA measure *percent determinism (DET)*:

$$DET = \frac{\sum_{l=d_{\min}}^{N} l\, H_{\mathrm{D}}(l)}{\sum_{i,j=1}^{N} r_{i,j}}. \tag{2}$$

It is the fraction of recurrence points that form diagonal lines; $H_{\mathrm{D}}(l)$ is the number of diagonal lines of exactly length l and $d_{\min}$ is a minimal length necessary to be a diagonal line. This measure characterizes the deterministic nature of a dynamical system from a heuristic point of view (further discussions can be found in [15, 19]). Further measures quantify average line lengths or the complexity of the line length frequency distributions $H_{\mathrm{D}}(l)$ (diagonal lines) and $H_{\mathrm{V}}(l)$ (vertical lines).

The time complexity of basic RQA measures is $O(N^2)$, where N denotes the number of data points. This property hampers an efficient computation for very long data. Furthermore, current implementations are limited by the memory these tools can manage. The CRP Toolbox for MATLAB [21] is limited to $N < 10,000$ data points when calculating the entire RP; for standard PC configurations even less, i.e., $N < 5,000$ data points. The RQA software by Webber Jr. [22] is capable of processing only up to $N = 5,000$ data points.

We present an algorithm based on the concept of *Divide and Recombine* (D&R) that allows RQA of very long time series in Sect. 2 and evaluate our algorithm in Sect. 3. In Sect. 4 we highlight the utilization of our approach for a concrete application example taken from the climate research domain.

2 Our Approach

2.1 Divide and Recombine

D&R is a very general approach to address large computational problems. The basic idea is to divide a data set into small sub sets allowing the fast computation of analytical results of the subsets. The intermediate results of the sub sets are recombined into a global solution.

The main application of D&R as presented by Guha et al. in [23] is to enable the analysis of big data sets. They use the *MapReduce* [24] framework to distribute the data and analytical computation between several computing nodes. In contrast, the central challenge in context of RQA is not the data volume itself but rather that the determination of RQA measures is compute intensive. To meet this challenge, we utilize the paradigm D&R as follows.

We divide the underlying matrix of a RP, the so called *recurrence matrix* (see Eq. (1)) into small sub matrices (*Divide*). For each sub matrix, we compute RQA measures in a massively parallel manner on GPU devices. This includes especially the detection of diagonal and vertical lines. The key issue of the divide step is distributing RQA computations of sub matrices between multiple GPU devices. In a final step, the individual results of the sub matrices are recombined into global RQA solution (*Recombine*).

Having distributed the computational load to several GPU devices using D&R, we further reduce the runtime by exploiting the parallel processing capabilities of a GPU device itself. We subdivide the processing within a sub matrix into a set of subtasks that can be processed concurrently. The underlying workflow of our approach is summarized in Fig. 1.

An important challenge with D&R for RQA is that diagonal and vertical lines may spread over a couple of sub matrices (see Fig. 2). To compute a valid global frequency distribution of diagonal and vertical line lengths, we introduce the *carryover buffer*. A carryover buffer is a data structure that allows to share information about the length of diagonal or vertical lines that exceed a sub matrix. In the following, we describe how our approach addresses the challenges of computing valid global frequency distributions of diagonal and vertical line lengths (see Sect. 1) as well as performing parts of the processing within a sub matrix in parallel.

2.2 Detection of Vertical and Diagonal Lines

To determine vertical lines within a recurrence matrix $\mathbf{r}$, the computation starts at element $r_{i,0}$, representing the first element of the i-th column. To capture the length of the current vertical line, we increase the line length counter if the element $r_{i,j}$ and $r_{i,j+1}$ are recurrence points (see Sect. 1). If $r_{i,j+1}$ is not a recurrence point, the vertical line stops at $r_{i,j}$. We then update the frequency distribution of vertical line

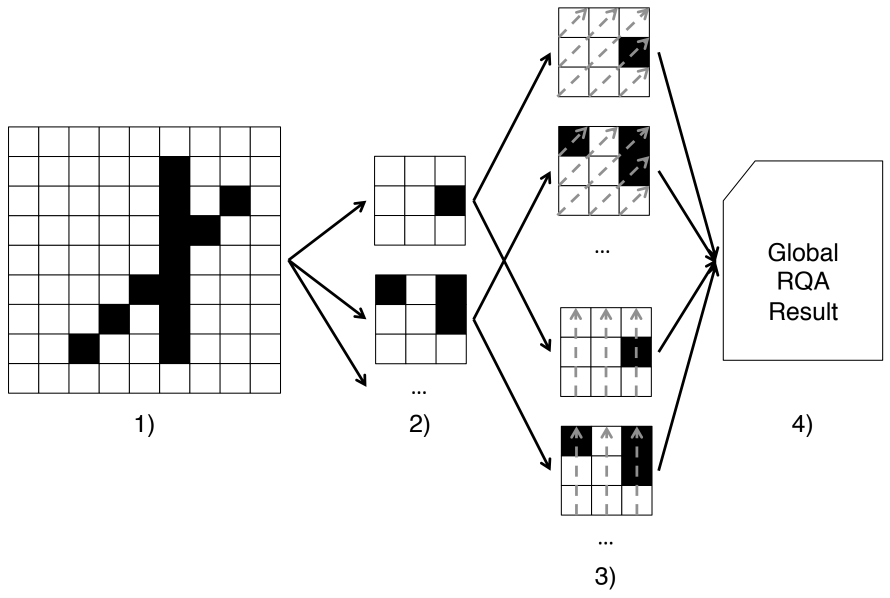
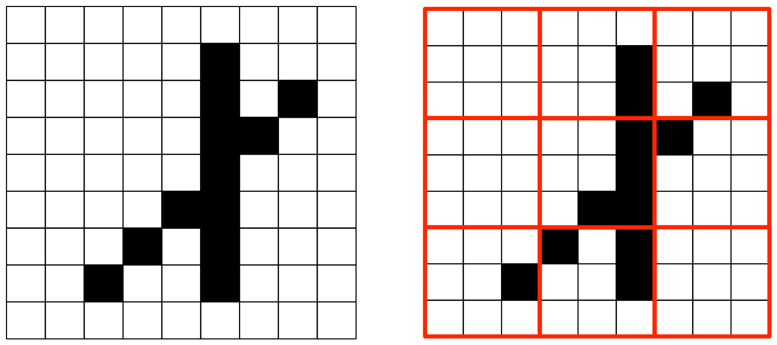


Fig. 1: Our D&R approach. (1) Given a recurrence matrix (2) we divide the recurrence matrix into a set of sub matrices. (3) We distribute sub matrices to several GPU devices. For each sub matrix, we compute the frequency distributions of diagonal and vertical lines. The computation of the frequency distributions of a single sub matrix is done in a massively parallel manner on a GPU device by identifying independent sub tasks; depicted as dotted arrows. (4) The individual results are recombined into a global RQA solution



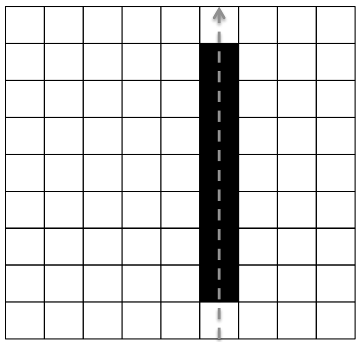
(a) Full Recurrence Plot (b) Divided Recurrence Plot

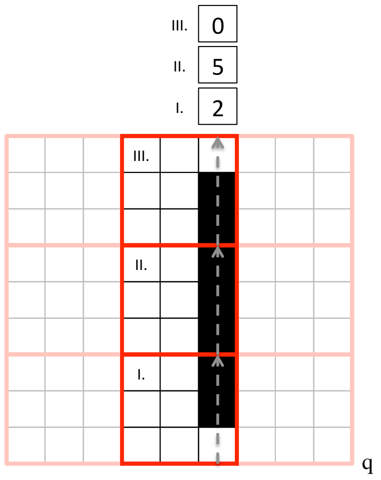
Fig. 2: Challenges of Divide and Recombine for RQA. The single diagonal and vertical line in (a) are distributed between several sub matrices in (b). Our approach computes valid global line lengths by utilizing carryover buffers.

lengths $H_\text{V}(l)$, set the line length counter to 0 and continue the detection of vertical lines at $r_{i,j+2}$. Note, each column of the recurrence matrix has a separate line length counter attached.

Subdividing **r**, its columns are split into a number of parts belonging to different sub matrices. We introduce the *vertical carryover buffer* to address this challenge. For each column of **r** it stores the length of a vertical line that exceeds the horizontal border of a sub matrix.

Fig. 3 compares the detection of vertical lines using the recurrence matrix as a whole (a) with applying the vertical carryover buffer to the set of sub matrices (b). The carryover buffer element of the column containing the single vertical line is shown above the recurrence plot. If a vertical line reaches the last element of a column of a sub matrix, the carryover buffer element stores its current length. Otherwise its value is 0.


(a) Full Recurrence Matrix



(b) Multiple Sub Matrices

Fig. 3: Detection of Vertical Lines. Detecting the vertical line in (a) is straightforward. Performing the processing on multiple sub matrices in (b) requires to preserve the execution order $I. \rightarrow II. \rightarrow III.$ The states of the carryover buffer element after processing each sub matrix (2, 5 and 0) is depicted on the top.

The value of the carryover buffer element is used as input for processing parts of the i-th column of **r** that belong to the adjacent sub matrix. To compute valid results, applying the carryover buffer requires a particular order of processing the sub matrices. We present the *vertical execution order rule* that reflects this dependency.

A sub matrix is referred to as $S_{g,h}$ with g being the horizontal and h being the vertical index. The sub matrix representing the bottom left corner of the recurrence matrix has the indices $g = 0$ and $h = 0$. The sub matrix representing the upper right corner is $S_{k,k}$, where k is the maximum value for g and h.

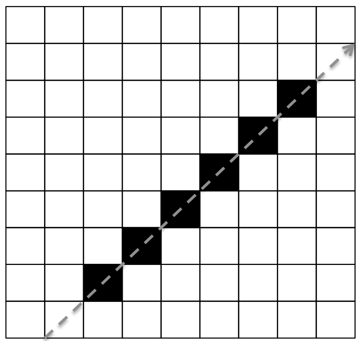
Vertical Execution Order Rule *Suppose a sub matrix $S_{g,h}$ with g being the horizontal index and h being the vertical index. All sub matrices $S_{g,m}$ with $0 \leq m < h$ have to be processed before $S_{g,h}$ is processed.*

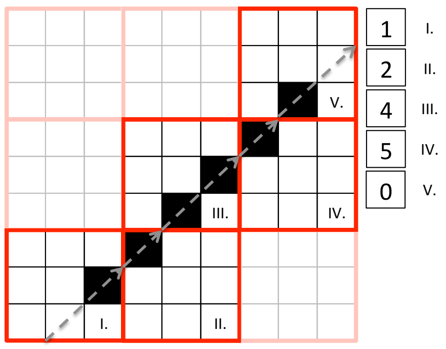
Sub matrices which do not share any element of the carryover buffer can be processed concurrently. This allows us to compute the frequency distribution of vertical line lengths of multiple sub matrices at the same time.

This concept can easily be adapted to the detection of diagonal lines, including the use of a carryover buffer and a particular order of execution concerning the set of sub matrices. The major difference is that a diagonal line may transcend not only the horizontal but also the vertical borders of sub matrices. Furthermore, the size of the carryover buffer is equivalent to the number of diagonals of $\mathbf{r}$.

Fig. 4 illustrates the detection of diagonal lines. For the purpose of demonstration, the RP contains only a single diagonal line of length 6. Since diagonal lines may cross horizontal as well as vertical sub matrix borders, we define a *diagonal execution order rule* that reflects this property.

Diagonal Execution Order Rule *Suppose a sub matrix $S_{g,h}$ with g being the horizontal index and h being the vertical index. All sub matrices $S_{m,n}$ that fulfill either the condition $(0 \leq n \leq g) \wedge (0 \leq m < h)$ or $(0 \leq n < g) \wedge (0 \leq m \leq h)$ have to be executed before $S_{g,h}$ is executed.*


(a) Full Recurrence Matrix



(b) Multiple Sub Matrices

Fig. 4: Detection of Diagonal Lines. Given a full recurrence matrix in (a), the detection of the diagonal line is straight-forward. By dividing the recurrence matrix in (b), the diagonal line stretches over 5 sub matrices. To preserve the validity of the detection result, they must be processed in the order $I. \rightarrow II. \rightarrow III. \rightarrow IV. \rightarrow V.$ The intermediate states of the corresponding carryover buffer element is depicted on the right of the matrix.

2.3 Parallel Processing within a Sub Matrix

Having distributed the processing of the set of sub matrices between a number of GPU devices, the RQA related computations within a single sub matrix offer additional potential for parallel processing.

Suppose a recurrence matrix of size $N \times N$. Each column comprises N matrix elements, which have to be processed in a specific order to identify vertical lines correctly. There are no dependencies concerning the execution order between matrix elements belonging to different verticals. Hence, by dividing the recurrence matrix into $K \times K$ sub matrices, for each sub matrix $\frac{N}{K}$ subtasks can be processed concurrently.

Referring to the detection of diagonal lines, the recurrence matrix comprises $2N-1$ diagonals. There are no dependencies between elements of different diagonals. By dividing $\mathbf{r}$ into $K \times K$ sub matrices, for each sub matrix there exist $2\frac{N}{K}-1$ parallel subtasks.

3 Performance Evaluation

To evaluate our approach, we compare the performance characteristics of a three RQA implementations. We contribute an implementation of our D&R approach using version 1.1 of the OpenCL general purpose GPU (GPGPU) programming language [25]. To fulfill the performance requirements, we use the low-level programming language C++ for implementing the host program.

The hardware setup of the experiment consists of a off-the-shelf desktop workstation, containing an Intel i5-3570 quad-core CPU at up to 3.80GHz and 16GB of main memory. It also includes a NVIDIA GeForce GTX 690 that provides two GPU processors running at up to 1.019GHz; each of them is supplied with 2GB of memory. In the context of GPGPU programming, each GPU processor is treated as a separate computing device. The workstation runs on a 64-bit version of *OpenSuse* 12.1 with version 4.2.1 of the *CUDA Toolkit*.

We repeat an experiment six times, from which only the latter five are considered for calculating the mean runtime. By excluding the first run, we want to minimize the influence of the hardware as well as the operating system. Measuring the runtime, we rely on the *chrono* package of the *Boost* library [26] with an accuracy up to one millisecond.

As stated in [27], it is important to compare optimized code that is running on the GPU to optimized CPU code. For this reason, we compare the massively parallel GPU implementation approach to two non-distributed, non-D&R CPU implementations. As a baseline we refer to a single-threaded C++ implementation. Additionally, we employ a parallel C++ implementation that is extended with *OpenMP* statements. It executes the detection of diagonal and vertical lines using multiple CPU threads.

Table 1 and Figure 5 compare the runtime of the three implementations based on a time series capturing the Sinus wave. We vary the time series between 20,000 and 200,000 data points. For each experiment we use the same parameters for reconstructing the states from the time series (see Sect. 1). Furthermore, we set the size of the sub matrices processed by a GPU processor to $20,000 \times 20,000$.

In all cases, our OpenCL implementation outperforms the multi-threaded OpenMP implementation (up to a factor > 5) and the single-threaded C++ implementation (up to a factor > 28) using only one GPU processor. Using both GPU processors available, the runtime can be reduced additionally up to 40%.

Table 1: Runtime for RQA calculation for time series of varying length. Parameters are embedding dimension $m = 2$ and delay $\tau = 2$.

Length	OpenCL (2x GPU)	OpenCL (1x GPU)	OpenMP	Single Thread
20,000	0.8 sec	0.8 sec	1.1 sec	5.6 sec
40,000	1.7 sec	1.8 sec	4.4 sec	22.3 sec
60,000	2.5 sec	3.1 sec	10.0 sec	50.2 sec
80,000	3.4 sec	4.6 sec	17.7 sec	1 min 29.2 sec
100,000	4.5 sec	6.3 sec	27.6 sec	2 min 19.5 sec
120,000	5.6 sec	8.4 sec	39.9 sec	3 min 21.0 sec
140,000	6.7 sec	10.7 sec	54.3 sec	4 min 33.2 sec
160,000	8.2 sec	13.3 sec	1 min 10.9 sec	5 min 56.8 sec
180,000	9.7 sec	16.2 sec	1 min 29.7 sec	7 min 30.9 sec
200,000	11.4 sec	19.3 sec	1 min 50.7 sec	9 min 16.8 sec

4 Application to Climate Data

In the following we will investigate the hourly temperature dynamics by RQA, which will be applied on a measurement record of hourly air temperature in Potsdam. This record is one of the longest, non-interrupted, hourly climate records of the world. In our analysis it is covering the period from 1893 until 2011 (although the measurements are still ongoing), resulting in 1,043,112 data points (Fig. 6(a)). For the period between 1893 and 1974, the warming trend of the annual mean temperature was 0.46 K per century, but after 1974 the trend rose to 3.4 K per century (Fig. 6(b)). In the following we will consider the full time period as well as the two periods 1893–1974 (718,776 data points) and 1975–2011 (324,336 data points) separately.

To study the short-term dynamics, we remove the annual trend (seasonal cycle) from the data by phase averaging, resulting in an anomaly temperature record. We use a time delay embedding of dimension $m = 5$ and delay $\tau = 3$, which have been found by false nearest neighbors approach for finding m [**?**] as well as a combined autocorrelation and visual recurrence plot inspection approach for finding an opti-

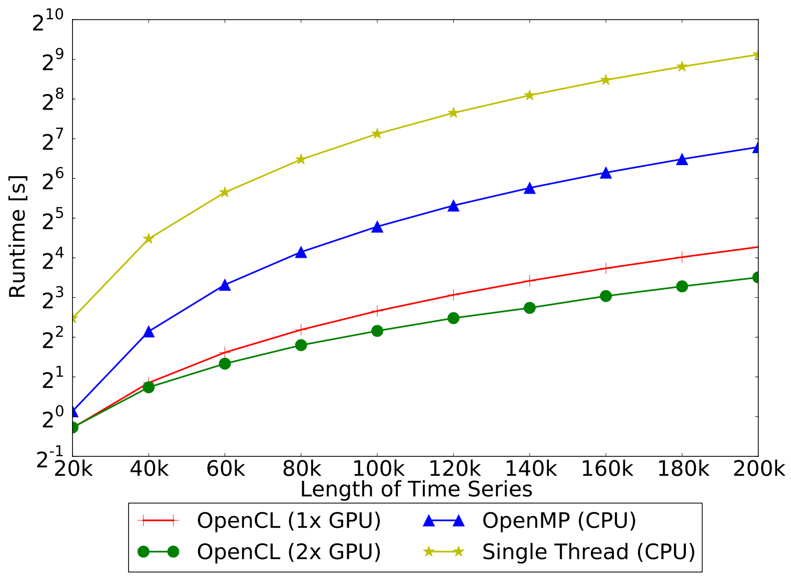


Fig. 5: Runtime Comparison. Our GPU implementation (OpenCL) of D&R outperforms both the single-threaded (C++) and multi-threaded (OpenMP) non-D&R CPU implementations. Balancing the work between two GPU processors, the runtime can be reduced additionally up to 40%.

mal τ [**?**]. We calculate the four RQA measures (1) recurrence rate RR, (2) determinism DET, (3) average diagonal line length L, and (4) laminartity LAM [15] for a recurrence threshold of $\varepsilon = 1$ (Euclidean norm). These measures reflect different aspects of the short-term dynamics, e.g., predictability. Due to the very long time series, we need to apply a highly parallelized computing procedure in order to receive results in a reasonable time. We find that all four measures do not remarkably change for the full period and the sub periods 1893–1974 and 1975–2011 (Tab. 2). This result suggests that, in contrast to the longer time-scales, the short-term dynamics, and, thus, the short-term weather predictability, has not (yet) changed due to the climate change.

The calculation of these RQA measures has benefited highly from our D&R approach, which made the calculations possible for these long time series. We applied the same implementations as well as the same experimental environment as described in Sect. 3. Using the single-threaded common RQA software, it takes over six hours to calculate the RQA for the full time period, whereas the OpenMP implementation still needs over one and a half hour. The OpenCL implementation allows to reduce the runtime to about five minutes, using two GPU processors (Tab. 3, Fig. 7). This significant runtime improvement of RQA will allow comprehensive investigations of big data collections of weather data, consisting of thousands of time series similar to the present Potsdam temperature record.

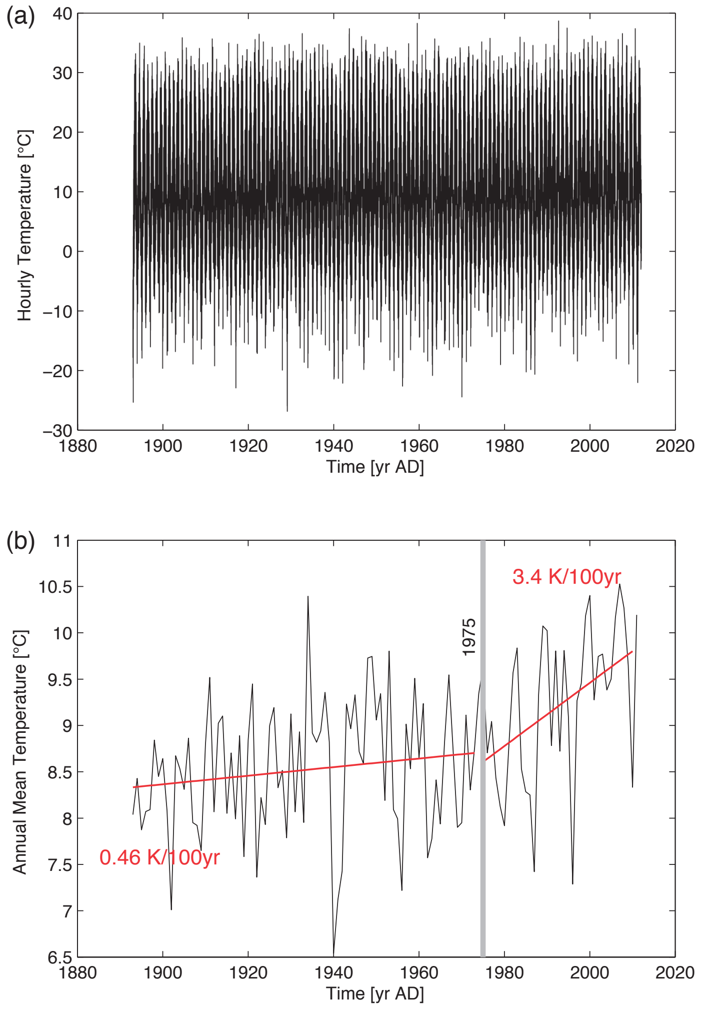


Fig. 6: (a) Hourly temperature in Potsdam, (b) annual mean temperature in Potsdam and warming trend for the periods 1893–1974 and 1975–2011.

Table 2: RQA results for the full time series of hourly temperature anomaly data of Potsdam as well for the two periods 1893–1974 and 1975–2011.

Measure	1893–2011	1893–1974	1975–2011
RR	0.12	0.12	0.13
DET	0.94	0.94	0.94
L	8.4	8.4	8.6
LAM	0.96	0.97	0.96

Table 3: Runtime for RQA calculation for the full time series of hourly temperature anomaly data of Potsdam as well for the two periods 1893–1974 and 1975–2011.

Computation schema	1893–1974	1975–2011	1893–2011
OpenCL (1x GPU)	4 min 40 sec	57 sec	10 min 11 sec
OpenCL (2x GPU)	2 min 25 sec	30 sec	5 min 10 sec
OpenMP (CPU)	44 min 38 sec	9 min 5 sec	1 h 33 min 58 sec
Single Thread (CPU)	2 h 59 min 25 sec	36 min 35 sec	6 h 18 min 4 sec

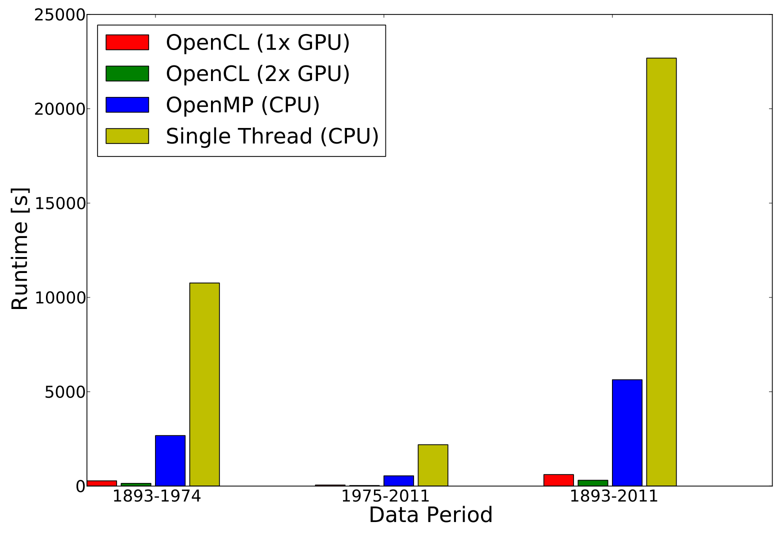


Fig. 7: Runtime for RQA calculation for the full time series of hourly temperature anomaly data of Potsdam as well for the two periods 1893–1974 and 1975–2011.

5 Conclusion

We present an approach based on the paradigm of D&R that allows to perform RQA on very long time series ($> 1,000,000$ data points) efficiently. By splitting the underlying matrix of a RP into a set of sub matrices, we are able to distribute the computational load between several GPU devices; each sub matrix is processed individually. We address the problem of diagonal and vertical lines transcending the borders of multiple sub matrices by introducing the diagonal and vertical carryover buffer as global data structures. To preserve the correctness of the RQA measures, we provide specific execution order rules for processing the individual sub matrices.

In comparison to a parallel CPU implementation using OpenMP, we can improve the runtime performance significantly up to factor > 9, using two GPU processors.

Our experiments have shown that GPU devices are well suited to compute basic RQA measures.

The application of the proposed RQA implementation to a specific problem from climate science has demonstrated its potential for an efficient recurrence study and will allow future RQA investigations of very long time series.

References

1. R.A. Muller, G.J. MacDonald, *Ice Ages and Astronomical Causes (Springer Praxis Books / Environmental Sciences)* (Springer, 2002)
2. D.I. Ponyavin, Solar Physics **224**(1–2), 465 (2005). DOI 10.1007/s11207-005-4979-5
3. J.F. Donges, R.V. Donner, M.H. Trauth, N. Marwan, H.J. Schellnhuber, J. Kurths, Proceedings of the National Academy of Sciences **108**(51), 20422 (2011). DOI 10.1073/pnas.1117052108
4. B. Goswami, N. Marwan, G. Feulner, J. Kurths, European Physical Journal – Special Topics **222**, 861 (2013). DOI 10.1140/epjst/e2013-01889-8
5. J.P. Zbilut, M. Koebbe, H. Loeb, G. Mayer-Kress, (IEEE Computer Society Press, 1990), pp. 263–266. DOI 10.1109/CIC.1990.144211
6. A. Porta, G. Baselli, N. Montano, T. Gnecchi-Ruscone, F. Lombardi, A. Malliani, S. Cerutti, Biological Cybernetics **75**(2), 163 (1996)
7. N. Marwan, N. Wessel, U. Meyerfeldt, A. Schirdewan, J. Kurths, Physical Review E **66**(2), 026702 (2002). DOI 10.1103/PhysRevE.66.026702
8. P. Van Leeuwen, D. Geue, M. Thiel, D. Cysarz, S. Lange, M.C. Romano, N. Wessel, J. Kurths, D.H.W. Grönemeyer, Proceedings of the National Academy of Sciences **106**(33), 13661 (2009). DOI 10.1073/pnas.0901049106
9. N. Marwan, Y. Zou, N. Wessel, M. Riedl, J. Kurths, Philosophical Transactions of the Royal Society A **371**(1997), 20110624 (2013). DOI 10.1098/rsta.2011.0624
10. S. Carrubba, C.F. II, A.L. Chesson Jr., A.A. Marino, Medicinal Engineering & Physics **32**(8), 898 (2010). DOI 10.1016/j.medengphy.2010.06.006
11. U.R. Acharya, S. Vinitha Sree, S. Chattopadhyay, W. Yu, P.C.A. Ang, International Journal of Neural Systems **21**(3), 199 (2011). DOI 10.1142/S0129065711002808
12. H. Bassily, J. Wagner, (2008), vol. 10, pp. 629–635
13. G. Litak, A.K. Sen, A. Syta, Chaos, Solitons & Fractals **41**(4), 2115 (2009). DOI 10.1016/j.chaos.2008.08.018
14. J. Iwaniec, T. Uhl, W.J. Staszewski, A. Klepka, Nonlinear Dynamics **70**(1), 125 (2012). DOI 10.1007/s11071-012-0436-9
15. N. Marwan, M.C. Romano, M. Thiel, J. Kurths, Physics Reports **438**(5–6), 237 (2007). DOI 10.1016/j.physrep.2006.11.001
16. N. Marwan, European Physical Journal – Special Topics **164**(1), 3 (2008). DOI 10.1140/epjst/e2008-00829-1
17. J.P. Zbilut, C.L. Webber Jr., Physics Letters A **171**(3–4), 199 (1992). DOI 10.1016/0375-9601(92)90426-M
18. C.L. Webber Jr., J.P. Zbilut, Journal of Applied Physiology **76**(2), 965 (1994)
19. N. Marwan, S. Schinkel, J. Kurths, Europhysics Letters **101**, 20007 (2013). DOI 10.1209/0295-5075/101/20007
20. K. Lehnertz, F. Mormann, T. Kreuz, R.G. Andrzejak, C. Rieke, P. David, C.E. Elger, IEEE Engineering in Medicine and Biology Magazine **22**, 57 (2003). DOI 10.1109/MEMB.2003.1191451
21. N. Marwan. CRP Toolbox 5.17 (2013). URL `http://tocsy.pik-potsdam.de/CRPtoolbox`. Platform independent (for Matlab)
22. C.L. Webber Jr. RQA Software 14.1 (2013). URL `http://homepages.luc.edu/\textasciitildecwebber`. Only for DOS

23. S. Guha, R. Hafen, J. Rounds, J. Xia, J. Li, B. Xi, W.S. Cleveland, Stat **1**(1), 53 (2012). DOI 10.1002/sta4.7. URL `http://dx.doi.org/10.1002/sta4.7`
24. J. Dean, S. Ghemawat, Commun. ACM **51**(1), 107 (2008)
25. *OpenCL 1.1 Specification* (2010). URL `http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf`
26. B. Dawes, D. Abrahams, R. Rivera. boost - C++ libraries (2013). URL `http://www.boost.org`
27. V.W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A.D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, P. Dubey, in *Proceedings of the 37th Annual International Symposium on Computer Architecture* (2010), ISCA '10, pp. 451–460