SP-TWDTW: A New Parallel Algorithm for Spatio-Temporal Analysis of Remote Sensing Images

Sávio S. T. de Oliveira[1], Luiz M. L. Pascoal[1], Laerte Ferreira[2], Marcelo de C. Cardoso[1], Elivelton Bueno[1], Vagner J. S. Rodrigues[1], Wellington S. Martins[1]

[1]Instituto de Informática - Universidade Federal de Goiás (UFG)
Alameda Palmeiras, Quadra D, Câmpus Samambaia
131 - CEP 74001-970 - Goiânia - GO - Brasil

[2]LAPIG - CAMPUS II Samambaia - Cx. POSTAL 131
CEP: 74001-970 - Goiânia - GO - Brasil

{savioteles,luizmlpascoal,lapig.ufg}@gmail.com,
{marcelo.cardoso,elivelton.bueno,vagner}@gogeo.io, wellington@inf.ufg.br

Abstract. *In the class of computationally complex problems, the time series analysis is one of those that has high demand for computational power. The Time-Weighted Dynamic Time Warping (TWDTW) algorithm stands out as one of the best solution found in the literature in this field, but its time complexity of $O(n^2)$ makes it unfeasible for large data sets. To overcome this limitation, this work proposes a parallel algorithm, named SP-TWDTW (Spatial Parallel TWDTW), that allows the analysis of large scale time series using Manycore architectures. The SP-TWDTW considers the temporal axis and the spatial autocorrelation to determine the land use mapping in a given region. The results show that the SP-TWDTW algorithm is a promising solution with response time up to 11 times lower.*

1. Introduction

The Earth's surface is changing at an unprecedented rate. Forest ecosystems diminish at alarming speed, urban and agricultural areas expand into the surrounding natural space. Since then, time series analysis of remote sensing images has become indispensable to identify these changes. It has attracted great interest in the world scenario, becoming an important resource in several applications [Kuenzer et al. 2015].

Among the group of time series analysis algorithms, the *Time-Weighted Dynamic Time Warping (TWDTW)* is considered one of the best algorithms for searching all possible occurrences of patterns in time series of remote sensing images [Maus et al. 2016]. The TWDTW algorithm is an adaptation of the *Dynamic Time Warping (DTW)* algorithm [Sakoe 1971], a well-known method for time series analysis. The DTW compares a pattern of a known event with an unknown time series.

Unlike the DTW, the TWDTW algorithm is sensitive to the seasonal changes of the natural and cultivated vegetation types, which is extremely important in remote sensing field [Maus et al. 2016]. However, the TWDTW analyzes each pixel individually, not taking into account the neighboring pixels and, therefore, make assumptions (e.g. independent, identical distributions) which violate Tobler's first law of Geography: everything is related to everything else but nearby things are more related than distant things (i.e.,

independent distributions). Techniques which ignore spatial autocorrelation typically perform poorly in the presence of spatial data [Vatsavai 2008].

This paper proposes a new highly parallel solution, named *Spatial Parallel TWDTW (SP-TWDTW)*, which takes into account the temporal axis and the spatial autocorrelation to determine the land use mapping in a given region. The TWDTW algorithm has a high computational cost, with time complexity of $O(n^2)$, which makes its use unfeasible for large data sets [Xiao et al. 2013] and makes it virtually impossible to analyze the spatial axis due to the computational cost. The SP-TWDTW explore the cores available in Manycore architectures and automatically manages the usage of the memory spaces to allows the processing of large amounts of data. The main contributions of this work are listed below:

- A new parallel algorithm for spatio-temporal analysis of remote sensing images.
- The inclusion of the spatial dimension in the classification of time series.
- An automatic system to manage memory usage between CPU and GPU spaces.

This paper is organized as follows. Section 2 discusses the processing of time series analysis for remote sensing images. Section 3 describes the TWDTW algorithm used as the basis for this work. The new algorithm proposed in this paper (SP-TWDTW) is presented in Section 4. Section 5 validates the SP-TWDTW algorithm experimentally and discusses the performance of the algorithm. Finally, Section 6 presents some conclusions and future work.

2. Analysis of Time Series for Remote Sensing Images

Time series analysis comprises methods for extracting important statistics and characteristics from time series data. The DTW, which is one of the most well-known methods in this field, allows the alignment between two time series, even if they have different lengths or they are not aligned on the time axis. Given the time series A and B, the distance between them are computed as

$$DTW(A,B) = min\sqrt{\sum_{k=1}^{K} w_k} \tag{1}$$

where $w_k = (i,j)$ represents the association between the i-th and the j-th observations, say a_i and b_j, respectively time series A and B, which are equivalents according to the Euclidean Distance, $d(i,j) = \sqrt{(a_i - b_j)^2}$. The sequence $w_1, w_2, ..., w_k$ represents the association between observation pairs of the two given time series, denoted by the adjustment path. Equation 1 is subject to the following conditions: i) The first observation of one series must match the first observation of the other series, $w_1 = (1,1)$, and the last observation of one series must match the last observation of the other, $w_k = (m,n)$; ii) Given $w_k = (i,j)$ and $w_{k+1} = (i',j')$ then $i' - i \leq 1$ and $j' - j \leq 1$; iii) Given $w_k = (i,j)$ and $w_{k+1} = (i',j')$ then $i' - i > 0$ and $j' - j \leq 0$. The DTW algorithm is not recommended for time series analysis of remote sensing images because it disregards the temporal range when finding the best alignment between two time series classification [Maus et al. 2016].

Some previous work like [Petitjean et al. 2012, Petitjean and Weber 2014, Maus et al. 2016] proposed non parallel methods using DTW to analyze time series of

satellite images, while [Petitjean et al. 2012, Petitjean and Weber 2014] used a maximum time delay to avoid time distortions based on the date of the satellite images. On the other hand, [Verbesselt et al. 2010, Jamali et al. 2015] support parallel execution in Multicore architectures using the library *foreach*[1] from R programming language. The TWDTW method [Maus et al. 2016] seeks to find all possible occurrences of a particular pattern within a time series, introducing time constraints, and has been prominent in the accuracy of identifying land cover use.

The rapid growth of multicore/manycore processors has attracted the attention of many researchers. For example, the works [Xiao et al. 2013, João Jr et al. 2017, Zhu et al. 2018] presented parallel solutions to analyze time series using many-core architectures (GPUs). However, they do not address the problems identified by [Maus et al. 2016] in the remote sensing field.

It is essential to make users aware of both the spatial and temporal dimensions in a Geographic Information System (GIS), since they may reveal implicit relationships which match the reality of the analyzed data [de Oliveira and de Souza Baptista 2012]. Several techniques of spatial time analysis have been previously proposed [Cressie and Wikle 2015]. Some methods process each image independently and compare the results for different time instances [Gómez et al. 2011, Lu et al. 2016]. The technique presented in [Costa et al. 2017] builds time series of each pixel and process them independently. At the end, the algorithm chooses some seed pixels in the image and calculates the distance between the time series of these seeds to their neighbors using the DTW method, grouping similar neighbors.

Some papers performs the time series analysis through spatial interpolation [Li and Heap 2014], which is the process of using points with known values to estimate values of other unknown points. Several methods are used, such as:

1. **Nearest Neighbor:** the value of each point is determined by the nearest points [Mitas and Mitasova 1999];
2. **IDW:** gives greater weights to points close to the prediction location [Shepard 1968];
3. **Kriging:** assumes that the distance or direction between sample points reflects a spatial correlation that can be used to explain variation in the surface [Stein 2012].

3. Time-Weighted Dynamic Time Warping (TWDTW)

The TWDTW [Maus et al. 2016] is a variation of the DTW algorithm that is sensitive to seasonal changes of natural and cultivated vegetation types. It considers inter-anual climatic and seasonal variability. The TWDTW method computes the cost matrix $\Psi_{n,m}$ given the pattern $U = (u_1, ..., u_n)$ and time series $V = (v_1, ..., v_m)$. The elements $\psi_{i,j}$ of $\Psi_{n,m}$ are computed by adding the temporal cost ω, becoming $\psi_{i,j} = |u_i - v_j| + \omega_{i,j}$, which $u_i \in U \; \forall \; i = 1, ..., n$ and $v_j \in V \; \forall \; j = 1, ..., m$. To calculate the time cost, the logistic model is used with a midpoint β and a bias α presented in Equation 2.

$$\omega_{i,j} = \frac{1}{1 + e^{-\alpha(g(t_i,t_j)-\beta)}}, \tag{2}$$

[1]https://cran.r-project.org/web/packages/foreach/index.html

in which $g(t_i, t_j)$ is the elapsed time in days between dates t_i for the patterns U and t_j in the time series V. From the cost matrix Ψ an accumulated cost matrix is calculated, named D by using a recursive sum of the minimum distances, as shown in equation 3

$$d_{i,j} = \psi_{i,j} + min\{d_{i-1,j}, d_{i-1,j-1}, d_{i,j-1}\}, \tag{3}$$

which is subject to the following conditions:

$$d_{ij} = \begin{cases} \psi_{i,j} & i = 1, j = 1 \\ \sum_{k=1}^{i} \psi_{k,j} & 1 < i \leq n, j = 1 \\ \sum_{k=1}^{j} \psi_{i,k} & i = 1, 1 < j \leq m \end{cases} \tag{4}$$

The kth lowest cost path in D produces an alignment between the pattern and a subsequence of V with associated distance δ_k, in which a_k is the first element and b_k the last element of k. Each minimum point in the last row of the cost matrix is accumulated, i.e. $d_{n,j} \; \forall \; j = 1, ..., m$, produces an alignment, with $b_k = argmin_k(d_{n,j}), k = 1, ..., K$ and $\delta_k = d_{n,b_k}$, in which K is the minimum number of points in the last row of D.

A reverse algorithm, equation 5, maps the path $P_k = (p_1, ..., p_L)$ along the kth "valley" to the lowest cost in D. The algorithm starts in $p_{l=L} = (i = n, j = b_k)$ and ends with $i = 1$, i.e. $p_{l=1} = (i = 1, j = a_k)$, in which L denotes the last point of alignment. The path P_k contains the elements that have been matched between the series.

$$p_{l-1} = \begin{cases} (i, a_k = j) & \text{se } i = 1 \\ (i - 1, j) & \text{se } j = 1 \\ argmin(d_{i-1,j}, d_{i-1,j-1}, d_{i,j-1}) & \text{otherwise} \end{cases} \tag{5}$$

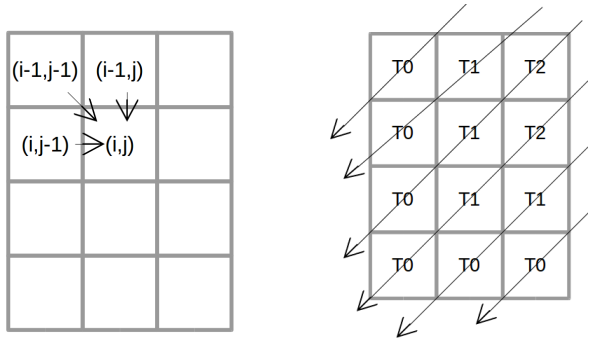
The land cover mapping with TWDTW is performed in two steps. In the first step, the DTW algorithm is applied to each pattern in $U \in Q$ and each time serie $V \in S$. This step provides information on how many patterns match with time series intervals. In the second step, the best matching pattern found by the DTW algorithm is used for land cover mapping.

4. Spatial-Time Series Analysis of Remote Sensing Images with a Parallel Architecture

The TWDTW is a pattern-matching algorithm based on dynamic programming with time complexity $O(n^2)$. This section presents the solution proposed in this work for the parallel processing of spatial time series analysis. This solution, named SP-TWDTW (Spatial Parallel TWDTW), parallelizes the TWDTW analyzing the temporal axis of the time series, as well as the spatial axis of the neighboring pixels to classify each time series.

The accumulated cost matrix D is computed from the cost matrix Ψ using the recursive sum of the minimum distances, as shown in equation 3. The construction of D can not be trivially paralleled since the computation of each element (i, j) of the matrix depends on the previously elements $(i-1, j)$, $(i, j-1)$ and $(i-1, j-1)$. This dependency can be seen in Figure 1(a). The idea behind the SP-TWDTW algorithm is presented in Figure 1(b). Each diagonal is computed in parallel, with each thread being responsible for

a diagonal cell. Since the elements are not dependent on each other within the diagonal, the calculation of the accumulated cost does not lead to an inconsistent matrix. The details of the SP-TWDTW matrix computation are presented in Algorithm 1 in Section 4.1.



(a) The computation of each element in D depends on the values of previous elements.

(b) SP-TWDTW: Parallel processing of D

Figure 1. Computation of the accumulated cost matrix D

4.1. Spatial Parallel Time-Weighted Dynamic Time Warping (SP-TWDTW)

Algorithm 1 describes the SP-TWDTW, which has as input the set of patterns Q and the set of time series S and calculates the final alignment cost matrix between each $U \in Q$ and $V \in S$. Since Q and S can be larger than available memory, the input of this algorithm admits that these sets are stored on the disk. The SP-TWDTW manages the loading of blocks of Q and S to CPU memory and subsequently to the GPU memory, so that it does not exceed the limits of them. The SP-TWDTW also receives as input the maximum size of the sets Q and S that fill in the GPU memory (bQ and bS respectively) and CPU memory (max_bQ and max_bS respectively).

The algorithm starts in the lines 2 and 3 by reading the blocks of the patterns into the $queueQ$ and blocks of the time series into the $queueS$. This work is performed by a CPU thread that manages the input queue size and the CPU memory available size. So, it is not necessary to wait to finish this step to start loading the blocks into the GPU global memory. Between lines 4 and 28, the Algorithm 1 loads the blocks into the GPU memory and computes the matrix D. This is done while there are blocks to be processed.

From line 5 until line 8, bQ patterns U and bS time series V are loaded into the GPU global memory, joining the patterns in a single $bigQ$ sequence and the time series in a single $bigS$ sequence. This union allows the GPU to perform block processing using all its computational power. In line 9, the cost matrix is constructed from $bigQ$ and $bigS$, with each GPU thread being responsible for computing an element of the array.

The calculation of matrix D is performed following the idea presented in Figure 1(b), where each thread computes the cost of each element in the diagonal. Since each element of the diagonal depends only on the two diagonals, they can be calculated independently. Given that $bigQ$ and $bigS$ have several patterns U and time series V, $bQ * bS$ threads are launched in the GPU, with each block of threads being responsible for calculating the cumulative cost between a pattern U and a time series V. Each block in the GPU

has $min(sizeU, sizeV)$ threads that is the maximum size of a diagonal when comparing U and V, so that each thread performs the calculation of one diagonal element.

From line 14 to 20, the costs of the first $sizeU$ elements from the upper diagonals above the secondary diagonal are computed. In a square matrix, these first $sizeU$ diagonals are the upper triangular matrix. The Algorithm 1 assumes that the index starts at position 0. To identify each diagonal, the row index in the upper matrix is computed as $si - tid$ and the column index is determined by the thread id. The matrix D is updated for each diagonal element using the function $update\_accumulated\_cost\_matrix$.

The elements of the next $sizeV - 1$ diagonals are computed between lines 21 and 26. In a square matrix, for example, these $sizeV - 1$ diagonals are the lower triangular matrix. To identify each diagonal, the row index in the upper matrix is calculated as $sizeU - tid - 1$ and the column index is set to $sizeV - sj - tid - 1$. The matrix D is then updated for each element of the diagonal.

The function $update\_accumulated\_cost\_matrix$ updates each element $D_{i,j}$ of the accumulated cost matrix D. The calculation of $D_{i,j}$ follows the equation 3, which is the smallest value between $D_{i-1,j}$, $D_{i-1,j-1}$ and $D_{i,j-1}$ plus the cost $\Psi_{i,j}$. The index in the matrices Ψ and D related to i, j of the matrices U and V must be computed, since the matrices Ψ and D are sent as vectors for the GPU and the series U and V in blocks of size bQ and bS respectively. So, it is necessary to find the pair U and V inside D and Ψ and then find the correct position in the matrix D related to U and V. In the GPU $bQ * bS$ blocks of threads are launched with bQ on the x-axis and bS on the y-axis. So, each block of threads handles one block in Ψ and D.

The matrix D is computed following equation 6 and the global index of Ψ and D follows the equation 7, in which $(i + blockIdx.x * sizeU) * sizeV * gridDim.y$ finds the correct line in Ψ and D, wherein $blockIdx.x$ the id of the block of threads in the x-axis and $gridDim.y$ the number of blocks in the y-axis. The part of the equation $(j + blockIdx.y * sizeV)$ finds within of the matrix line the correct position of the element (i, j), being $blockIdx.y$ the id of the block of threads in the y-axis.

$$D[index_{i,j}] = min(D[index_{i-1,j}], D[index_{i-1,j-1}], D[index_{i,j-1}) + \Psi[index_{i,j}] \quad (6)$$

$$index = (i + blockIdx.x * sizeU) * sizeV * gridDim.y + (j + blockIdx.y * sizeV) \quad (7)$$

The algorithm computes the final alignment between each U and V following the equation 5 in the function $compute\_dtw\_path$ between lines 29 and 31. This cost is stored in the matrix R, which each row represents a pattern U and each column a time series V to be sorted.

Between lines 34 to 39, the SP-TWDTW analyzes the spatial axis, performing a spatial interpolation on line 36 for each alignment between U and V, that is, each element of R. We already have the cost of the alignment between U and V stored in R_{ij}, but in the line 36, the method $compute\_spatial\_interpolation(R_{i,j})$ searches for the cost of other spatially close time series to V related to the pattern U, estimate the value of R_{ij} and stores it in the matrix I at I_{ij}. There are several factors that impact on the quality of this spatial prediction [Li and Heap 2014] and, therefore, a mechanism has been made where it is possible to give weights for temporal and spatial axis in line 37 of the algorithm. This weighted cost is stored in the output R matrix.

Algorithm 1: sp-twdtw(Q, bQ, max_bQ, S, bS, max_bS, $temporal\_weight$, $spatial\_weight$)

Input: Q: set of patterns U
bQ: number of patterns U in the GPU memory
max_bQ: max patterns U in the CPU memory
S: set of time series V
bS: number of time series V in the GPU memory
max_bS: max time series V in the CPU memory
$temporal\_weight$: temporal axis weight
$spatial\_weight$: spatial axis weight
Output: R: final alignment cost matrix between each U and V

```
while There are patterns U and time series V on disk do
    queueQ ← load max_bQ patterns U to CPU memory
    queueS ← load max_bS timeseries V to CPU memory
    while There are patterns in queueQ and time series in queueS do
        gpu_queueQ ← load bQ patterns U to GPU global memory
        gpu_queueS ← load bS time series V to GPU global memory
        bigQ ← merge all patterns U in gpu_queueQ
        bigS ← merge all time series V in gpu_queueS
        Ψ ← compute the cost matrix between bigQ and bigS
        sizeU ← compute the pattern size of each U in bigQ
        sizeV ← compute the time series size of each V in bigS
        tid ← thread id
        if tid < sizeU then
            for si ← 0 to sizeU - 1 do
                if tid ≤ min(si, sizeV − 1) then
                    i ← si − tid
                    j ← tid
                    update_accumulated_cost_matrix(Ψ, D, i, j, sizeU, sizeV)
                end
            end
            for sj ← sizeV − 2 to 0 do
                if tid ≤ min(sj, sizeU − 1) then
                    i ← sizeU − tid − 1
                    j ← sizeV − sj − tid − 1
                    update_accumulated_cost_matrix(Ψ, D, i, j, sizeU, sizeV)
                end
            end
        end
        for Each U in gpu_queueQ and V in gpu_queueS do
            compute_dtw_path(R, D, sizeU, sizeV)
        end
    end
end
for Each line i in R do
    for Each column j in R do
        I_{i,j} ← compute_spatial_interpolation(R_{i,j})
        R_{i,j} ← R_{i,j} * temporal_weight + I_{i,j} * spatial_weight
    end
end
```

The drawback of the diagonal based method is that the sizes of the diagonals vary, which causes a waste of GPU resources. When the diagonal size is lower than the number of block threads, some of these threads become idle. But, the performance gain in parallelizing the computation of the diagonal is better than sequential computation.

5. Experiments and Results

This section aims to evaluate the performance of the SP-TWDTW and TWDTW algorithms executed on Multicore (CPU) and Manycore (GPU) architectures. In the CPU tests, a machine with AMD FX-8320E (3.2 GHz, 8 MB Cache) and 8 GB of RAM was used. The GPU tests were performed on a NVIDIA GeForce GTX 1050 Ti card with 4 GB GDDR5 of available memory and 768 CUDA cores with clock of 1392 MHz. The TWDTW was implemented in C++ [2] and SP-TWDTW was implemented on the GPU using the NVIDIA CUDA language. To compare the response time between the SP-TWDTW and the TWDTW, the time series V and the patterns U were obtained from real data in Brazil [Maus et al. 2016] from MODIS sensor, specifically the Porto dos Gauchos municipality, that covers approximately 7,000 km^2 and is located in the state of Mato Grosso, Brazil, inside of the Amazon Biome. Each test was performed ten times and the response time was obtained from the average of them.

Temporal and spatial resolution of remote sensing system have increased in the last years being a great challenge for remote sensing field [Battude et al. 2016]. The results regarding the response time for high temporal resolution is presented in Figure 2(a), by comparing the TWDTW and SP-TWDTW over a pattern U and a time series V, varying their size. The response time increases considerably with TWDTW as the size of the pattern U and the time series V increases.

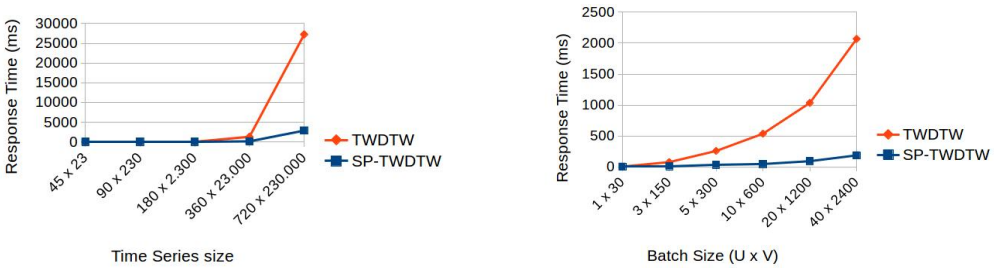
As one can see in Figure 2(a), the TWDTW algorithm works better for smaller time series (e.g. less than or equal to 90 x 230), however, when the size exceeds 90 x 230 it becomes more advantageous to use the SP-TWDTW. This is due to the fact that SP-TWDTW uses the GPU for processing and, in this case, the transfer time of U and V from the CPU memory to GPU memory overcomes the final response time of the algorithm. The SP-TWDTW algorithm presented response time up to 10 times lower than TWDTW.

Next, the results regarding the high resolution on spatial data are shown in Figure 2(b) by comparing the response time of SP-TWDTW and TWDTW using small batches of time series U and V with size of 45 and 23 respectively. As the batch size increases the difference in response time also increases. For batch size of 3x150, the response time of the SP-TWDTW was 7 times lower than the TWDTW. The SP-TWDTW used a batch size equal to 10x600, running several batches to calculate the alignment between each U and V. In this scenario, the SP-TWDTW obtained response time 11 times lower than the TWDTW, since it explored better the GPU architecture.

To compare the accuracy of the SP-TWDTW algorithm and TWDTW algorithm, we selected the Porto dos Gauchos municipality, located in the state of Mato Grosso, Brazil, inside of the Amazon Biome. Data from this study area were obtained from TWDTW's github [3] and covers approximately 7000 km^2 with 541 time series. The same

[2]The original version of TWDTW was developed in R, but, in this work, we implemented in C++ to be able to compare fairly with SP-TWDTW, since the C++ language has better performance.

[3]https://github.com/vwmaus/dtwSat



(a) Comparison between the TWDTW and the SP-TWDTW algorithms using only one pattern U and one time series V varying their size. The x axis contains the size of U and V following the format: $sizeofU$ x $sizeofV$.

(b) Comparison between the TWDTW and the SP-TWDTW algorithms with several patterns U and time series V. The x axis follows the format: number of patterns U x number of time series V. Each pattern has size 45 and time series size 23.

Figure 2. Comparing the response time between SP-TWDTW and TWDTW.

value of α and β were use for both methods. The Kriging, IDW and Nearest Neighbor(NN) spatial interpolation methods were used, with their parameters have been optimized through cross validation.

The Table 1 presents the results for time series classification using the SP-TWDTW. An analysis was performed by varying spatial and temporal axis weights and the interpolation methods. The SP-TWDTW algorithm with weight 0 for the spatial axis and 1 for temporal shows the number of time series incorrectly classified by the TWDTW. It is noteworthy that, in this case, the SP-TWDTW brings the gain of response time reduction, as presented in Figure 2, but not in accuracy. The original TWDTW algorithm implemented in R [Maus et al. 2016] obtained a response time of 9851 ms while the SP-TWDTW on the GPU took 40 ms, that is, 246 times faster than the TWDTW in R.

Regardless of the interpolation method, when the spatial axis are set with higher value of weight than the time axis, the SP-TWDTW presented lower accuracy than the TWDTW. This is explained on Table 2 by spatial closeness between the “Soybean-maize”, “Soybean-cotton” and “Cotton-fallow” crops. For “Forest” and “Soybean-millet” land usage, the SP-TWDTW didn’t misses even with spatial weight equals to 1 because the data in these land usage are clustered and independent from others land classes. Concerning to interpolation methods, the IDW performed better that the other methods due to the uniform data distribution. For sparse data, the Kriging method would probably present better accuracy than the IDW [Li and Heap 2014].

Table 2 presents the accuracy assessment for each land usage type according to the TWDTW and SP-TWDTW time series analysis methods and the IDW spatial interpolation method. The spatial and temporal weights were fixed on 0.4 and 0.6 respectively. The accuracy for SP-TWDTW was equal or better for all land use types, improving the TWDTW accuracy in a region that it already presented great results. Since the “Soybean-maize”, “Soybean-cotton” and “Cotton-fallow” crops are spatially closed and mixed, the SP-TWDTW negatively impacted on the analysis of the spatial axis.

Finally, in the results for the established scenarios, the SP-TWDTW presented a response time up to 11 times lower than the TWDTW in $C++$ and 246 times lower than

Weights		Interpolation Methods		
Spatial	**Temporal**	**Kriging**	**NN**	**IDW**
0	1	9	9	9
0,1	0,9	8	8	8
0,2	0,8	8	8	8
0,3	0,7	7	8	7
0,4	0,6	8	8	7
0,5	0,5	8	10	8
0,6	0,4	12	12	10
0,7	0,3	12	15	12
0,8	0,2	14	15	12
0,9	0,1	14	15	12
1	0	14	18	15

Table 1. Number of samples that were incorrectly classified by the SP-TWDTW method varying the spatial and temporal weights and the interpolation methods.

Method	Cotton-fallow		Forest		Soybean-cotton		Soybean-maize		Soybean-millet	
	UA (%)	PA (%)	UA (%)	PA (%)	UA (%)	PA (%)	UA (%)	PA (%)	UA (%)	PA (%)
TWDTW	95	100	100	100	100	87	95	100	100	100
SP-TWDTW	96	100	100	100	100	90	97	100	100	100

Table 2. Accuracy evaluation for each land usage type according to the TWDTW and SP-TWDTW algorithms.

the TWDTW in R, presenting a viable alternative to the time series analysis in the Big Remote Sensing Data era. The SP-TWDTW also presented similar user's and producer's accuracy than the traditional TWDTW.

6. Conclusion

In the class of complex computational problems, the time series analysis is one of the problems with increased demand for computing power [Rakthanmanon et al. 2013], due to the complexity of the algorithms and the large volume of data to be processed.

The TWDTW algorithm has been highlighted as one of the best solution found in the literature to perform the time series analysis for remote sensing images. However, TWDTW disregards the first law of Geography, processing each pixel independently. In addition, the TWDTW algorithm presents time complexity of $O(n^2)$, becoming unfeasible for large data sets.

This work presented a parallel solution capable of analyzing large volumes of time series data exploring parallel architectures. The solution, named SP-TWDTW (Spatial Parallel TWDTW), analyzes the spatial and temporal dimensions using the TWDTW algorithm to temporal analysis and the interpolation methods to spatial analysis. To support the Big Remote Sensing Data scenario, the SP-TWDTW took advantage of the Manycore architecture with the coordinated and appropriate usage of the large number of available cores. The SP-TWDTW processes the time series as batches, managing the CPU and GPU memory spaces to allows the processing of large amount of data without exceeding their capacity.

As results, the SP-TWDTW proved to be a promising solution for high temporal resolution data, with a speedup of 10 times over the traditional TWDTW and almost 11 times less response time compared to TWDTW for high spatial resolution data. Using spatial interpolation methods, SP-TWDTW was able to increase the accuracy of TWDTW for land use mapping in the Amazon region.

As future work, we intend to evaluate the SP-TWDTW algorithm using large data sets. We also intend to integrate the SP-TWDTW algorithm into the *DistSensing* platform [de Oliveira et al. 2017] exploring the resources of a cluster of computers.

References

Battude, M., Al Bitar, A., Morin, D., Cros, J., Huc, M., Sicre, C. M., Le Dantec, V., and Demarez, V. (2016). Estimating maize biomass and yield over large areas using high spatial and temporal resolution sentinel-2 like remote sensing data. *Remote Sensing of Environment*, 184:668–681.

Costa, W. S., Fonseca, L. M., Körting, T. S., SIMÕES, M., Bendini, H. N., and Souza, R. C. (2017). Segmentation of optical remote sensing images for detecting homogeneous regions in space and time. In *Embrapa Solos-Artigo em anais de congresso (ALICE)*. In: BRAZILIAN SYMPOSIUM ON GEOINFORMATICS, 18., 2017, Salvador. Proceedings... Salvador: Unifacs, 2017. p 40-51.

Cressie, N. and Wikle, C. K. (2015). *Statistics for spatio-temporal data*. John Wiley & Sons.

de Oliveira, M. G. and de Souza Baptista, C. (2012). Geostat-a system for visualization, analysis and clustering of distributed spatiotemporal data. In *GeoInfo*, pages 108–119.

de Oliveira, S. S. T., de Castro Cardoso, M., dos Santos, W., Costa, P., do Sacramento Rodrigues, V. J., and Martins, W. S. (2017). Distsensing: A new platform for time series processing in a distributed computing environment. *Revista Brasileira de Cartografia*, 69(5).

Gómez, C., White, J. C., and Wulder, M. A. (2011). Characterizing the state and processes of change in a dynamic forest environment using hierarchical spatio-temporal segmentation. *Remote Sensing of Environment*, 115(7):1665–1679.

Jamali, S., Jönsson, P., Eklundh, L., Ardö, J., and Seaquist, J. (2015). Detecting changes in vegetation trends using time series segmentation. *Remote Sensing of Environment*, 156:182–195.

João Jr, M., Sena, A. C., and Rebello, V. E. (2017). Implementação e avaliação de técnicas de paralelização no algoritmo de hirschberg para sistemas multicore. *Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)*.

Kuenzer, C., Dech, S., and Wagner, W. (2015). Remote sensing time series revealing land surface dynamics: Status quo and the pathway ahead. In *Remote Sensing Time Series*, pages 1–24. Springer.

Li, J. and Heap, A. D. (2014). Spatial interpolation methods applied in the environmental sciences: A review. *Environmental Modelling & Software*, 53:173–189.

Lu, M., Chen, J., Tang, H., Rao, Y., Yang, P., and Wu, W. (2016). Land cover change detection by integrating object-based data blending model of landsat and modis. *Remote Sensing of Environment*, 184:374–386.

Maus, V., Câmara, G., Cartaxo, R., Sanchez, A., Ramos, F. M., and de Queiroz, G. R. (2016). A time-weighted dynamic time warping method for land-use and land-cover mapping. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(8):3729–3739.

Mitas, L. and Mitasova, H. (1999). Spatial interpolation. *Geographical information systems: principles, techniques, management and applications*, 1:481–492.

Petitjean, F., Inglada, J., and Gançarski, P. (2012). Satellite image time series analysis under time warping. *IEEE Transactions on Geoscience and Remote Sensing*, 50(8):3081–3095.

Petitjean, F. and Weber, J. (2014). Efficient satellite image time series analysis under time warping. *Ieee geoscience and remote sensing letters*, 11(6):1143–1147.

Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., and Keogh, E. (2013). Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 7(3):10.

Sakoe, H. (1971). Dynamic-programming approach to continuous speech recognition. In *1971 Proc. the International Congress of Acoustics, Budapest*.

Shepard, D. (1968). A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524. ACM.

Stein, M. L. (2012). *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media.

Vatsavai, R. R. (2008). *Machine Learning Algorithms for Spatio-temporal Data Mining*. PhD thesis, University of Minnesota, Minneapolis, MN, USA. AAI3338985.

Verbesselt, J., Hyndman, R., Newnham, G., and Culvenor, D. (2010). Detecting trend and seasonal changes in satellite image time series. *Remote sensing of Environment*, 114(1):106–115.

Xiao, L., Zheng, Y., Tang, W., Yao, G., and Ruan, L. (2013). Parallelizing dynamic time warping algorithm using prefix computations on gpu. In *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*, pages 294–299. IEEE.

Zhu, H., Gu, Z., Zhao, H., Chen, K., Li, C.-T., and He, L. (2018). Developing a pattern discovery method in time series data and its gpu acceleration. *Big Data Mining and Analytics*, 1(4).