cuTimeWarp: Accelerating Soft Dynamic Time Warping on GPU

Alex Kyllo Afrooz Rahmati

March 15, 2021

Abstract

This report explores techniques for optimizing the computation of Soft Dynamic Time Warping, a differentiable sequence dissimilarity measure, on graphics processing units (GPU), for the purpose of enabling high-performance machine learning on time series datasets.

Introduction

Time series machine learning is a research area with countless useful applications such as recognizing sounds and gestures. Clustering or classifying large time series datasets is challenging partly because of the need to define a measure of dissimilarity between any two given time series. Furthermore, practical applications require finding common structure in time series despite different speeds or phases; for example, a word means the same whether spoken quickly or slowly. Another requirement for machine learning tasks is that the dissimilarity measure must be differentiable so that its gradient can be used as to minimize it as a loss function to find the best fit model. Finally, the measure must be efficient to calculate, because it will be calculated repeatedly many times during model fitting. To this end we will explore GPU acceleration of Soft Dynamic Time Warping (Soft-DTW) [1], a differentiable sequence dissimilarity measure, to enable high performance time series machine learning.

Background

Time series data

Time series data generally refers to data containing quantitative measurements collected from the same subject or process at multiple points in time, and it exhibits several peculiarities in comparison to time-independent data. Time series data can exhibit autocorrelation, meaning that the value at any point in time can exhibit correlation to the value at a previous point in time, with some delay or *lag* in between. Time series data can also exhibit *seasonality* or cyclical patterns as well as *trend* which is a tendency for the average value (over some rolling time window) to increase or decrease as a function of time. Due to these characteristics, time series data does not conform to the I.I.D. (independent and identically distributed) assumption that typically applies in the study of random variables, and therefore it requires special techniques for analysis and modeling.

Time series data can be either univariate or multivariate. For example, a set of electrocardiogram (ECG) measurements has a single variable (heart voltage) collected over time, but if the dataset also

included the patient's blood pressure and oxygen levels measured at each time point, that would constitute a multivariate time series, and correlations between the different variables could be studied in addition to the autocorrelation within each of the variables.

Time series distance and dissimilarity measures

A fundamental capability that enables learning from data is the ability to quantify a metric of distance or dissimilarity between observations, because this allows comparison among data observations as well as quantification of error between observed data and the predictions of an estimator model. For time-independent data, multiple valid notions of distance exist; the most commonly used is Euclidean distance, but others such as Manhattan distance, cosine distance, or Mahalanobis distance are often used depending on the problem domain.

Likewise, there are multiple valid ways to compute a measure of dissimliarity between two sequences or time series; for real-valued time series measurements the simplest is also Euclidean distance, which is the square root of the sum of squared pairwise differences between two time series x and y, each of length n (equation 1).

$$d(x, y) = \sqrt{\sum_{t=1}^{n} (x_t - y_t)^2} \tag{1}$$

However, a significant drawback of Euclidean distance in time series applications is that two structurally similar time series will produce a large distance if they are at different speeds or out of phase (TODO: Add a figure to illustrate this). In time series applications it is often desirable to produce a low dissimilarity for structurally similar time series despite variations in phase or speed, so an alternative method of quantifying dissimilarity is needed.

Dynamic Time Warping

Dynamic Time Warping (DTW) was devised in the 1960s as an alternative time series dissimilarity measure to address this shortcoming. (TODO: find a historical citation). DTW is a nonlinear mapping of from each point in one time series to the nearest point in a second time series. While DTW is technically not considered a "distance" because it does not conform to the triangle inequality, and therefore we refer to it as a "dissimilarity" instead, it can be used in place of Euclidean distance or other distance measures for many applications of time series data.

The purpose of (DTW) is to perform a transformation function that wrap the time to align two time series. Preferably we tend this alignment to be optimum and satisfy our requirements.[2]

DTW is a widely used tool employed in many areas of science, including biology, technology, economics, and finance. It calculates the practical distance between two signals typically by taking the distance between them when one is time warped, or it can be the minimum wrapping required to align signals with each other by applying some sort of fidelity. Dynamic time wrapping can be used to identify hidden pattern or searching within signals databases to find the matching one.[3] it is utilized in machine learning platforms that depends on signals like clustering, regression, and classification.

The basic algorithm for DTW is to use Bellman's recursion, a dynamic programming technique, to find the lowest-cost path diagonally across a pairwise distance matrix. The computation cost for this approach is quadratic ($O(mn)$) for time series vectors of length m and n [1]. The formula for the DTW

between time series x and y is given by equation 2. (TODO: explain the algorithm in more detail and provide a visualization)

$$DTW(x,y) = min_{\pi} \sqrt{\sum_{(i,j)\in\pi} d(x_i, y_j)^2} \tag{2}$$

Where $d(x_i, y_j)^2$ is the cost function, typically pairwise squared Euclidean distance. The loss function for DTW is not differentiable due to the min operation within the formula; a small change in the input time series may result in zero change in the path cost. However, we can create a differentiable version called Soft-DTW by replacing the min function with a soft-min function (equation 3) [1].

$$\text{soft-min}_{\gamma}(a_1, ..., a_n) = -\gamma log \sum_i e^{-a_i/\gamma} \tag{3}$$

Hence, Soft-DTW is parameterized by the smoothing constant gamma, which becomes a tunable hyperparameter in machine learning model training applications.

Applications

As a differentiable time series dissimilarity measure, DTW can be applied as a loss function or minimization objective in data modeling techniques such as:

- k-means Clustering
- Hierarchical agglomerative clustering
- Motif (similar subsequence) search
- k-nearest neighbor or nearest centroid classification
- Recurrent neural networks

A common technique in machine learning with Soft-DTW is the computation of barycenters, which are centroids within the space of a set of time series. The differentiability of Soft-DTW allows for barycenter finding via gradient descent, and then new observations can be classified by finding the nearest barycenter. Sequence prediction and generation is also possible using recurrent neural networks with Soft-DTW as a loss function [1].

(TODO: Explain process of barycenter finding with gradient descent on softdtw loss)

Prior to computing the Soft-DTW dissimilarity between any two time series, each time series should be *z-normalized*, that is, scaled so that its mean is equal to 0 and its standard deviation is equal to 1, to remove the problem of "wandering baselines" or "drift" in the measurements, as illustrated in [4] with an ECG classifier that yields incorrect results on un-normalized data due to drift that "may be caused by patient movement, dirty lead wires/electrodes, loose electrodes, and so on," and which "does not have any medical significance." Z-normalization also tends to make the iterative process of minimizing the cost function through gradient descent or quasi-Newtonian methods more efficient because its hyperplane is not disproportionately stretched in any one dimension, so the step size in any direction is the same relative to the scale of that dimension. (TODO: explain this better, find a citation)

Parallelizing DTW

A naive, sequential implementation of DTW would involve a nested loop to iterate over each row/column of the cost matrix to update its cost based on the three neighboring cells' costs, hence the $O(n^2)$ time complexity. But because each cell has a data dependency on the three cells to the top, left, and top-left, there is no dependency between cells that are on the same antidiagonal of the matrix, therefore computation of these cells can be handled by parallel threads. One thread computes the upper-leftmost cell, then two threads compute the next antidiagonal, then three threads compute the next, and so on.

Related Work

Utilizing indexing to construct lower bounds on warping distance is an optimization technique for speeding up nearest neighbor search via early removal of poor candidates [5]. Shen and Chi (2021) proposes an optimization of nearest neighbor search of multivariate time series, leveraging the triangle inequality and quantization-based point clustering to restrict the search [6].

Xiao et al (2013) introduced a prefix computation technique for transforming the diagonal data dependence to improve parallel computation of the cost matrix on GPU [7]. Zhu et al (2018) demonstrates a method of optimizing memory access by taking advantage of the diagonal data dependency to rearrange the matrix so that elements on the same diagonal are stored contiguously [8]. A prior implementation of Soft-DTW on CUDA using PyTorch and Numba is capable of 100x performance improvement over the original Soft-DTW Cython code, but is limited to sequence lengths of 1024 (CUDA max block size) and leaves many opportunities for further CUDA optimizations such as the use of shared memory [9]. A 2015 paper describes a tiling approach called *PeerWave*, which utilizes direct synchronization between neighboring streaming multiprocessors (SMs) to handle the inter-tile data dependency without atomic operations, locks, or other global barriers, leading to improved load balance and scaling properties [10].

In our project we will focus on this area of opportunity, optimizing matrix structure and memory access patterns to maximize parallelism and minimize memory latency in the computation of the warping path matrix.

Methods

To evaluate various performance optimizations on the Soft-DTW computation, we implemented a C++ and CUDA library called cuTimeWarp, which includes functions for computing the SoftDTW on CPU and GPU.

Given a set of many multivariate time series of the same length and number of variables, we can compute the Soft-DTW distance between every time series and every other time series in the set by computing, in parallel for each pair, a pairwise squared Euclidian matrix, then applying the Soft-DTW calculation on the distance matrix. This computation, however, also has an $O(n^2)$ complexity and can potentially even take longer than the DTW computation itself. For univariate time series, we can save this cost by computing the DTW cost matrix on the two input time series directly, computing the absolute distance between each pair of values from the two time series within the nested loop of the DTW procedure.

Optimization Techniques

Wavefront Tiling

Wavefront parallelism is one of the useful methods to overcome the dependencies of nested loops by multiple processing units. The idea is to re-order the loop iterations in such a way that form diagonal wave and each wave can be computed in parallel. Barriers will be utilized to control the data dependencies among consecutive waves. Elements inside the wave grouped together using tiled technique to enhance data locality and performance. This methodology presents a second degree of parallelism where tiles can be computed in parallel by separating with a global variable.

GPU and specifically CUDA can accelerate the process of wavefront. Each tile assigns to a block to be process in parallel by SM. On the other hand, the iteration along diagonals within a tile are also pluralized. In order to enforce the dependencies, the global barriers used among the tiles and within every tiles.[10]

In our Soft DTW implementation, we utilized the wavefront technique to manage the dependencies of neighboring cells for computing the minimum warp path. In our algorithm this value depends on the minimum cost of the upper, left and upper-left diagonal cell cost. Figure 1 show this dependencies clearly.Each D(i,j) depends on the up, left and up-left neighbor cost.

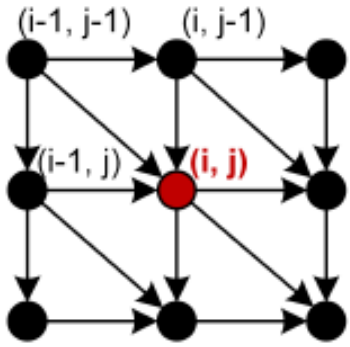

Figure 1: computation dependencies for DTW.

Wavefront process split to three major steps:

- Populating the dependencies managing by loop. In this step, we keep the global variable WaveId to sperate the process of waves. Wavelength is increasing one by one on each loop iterations until reaching the total number of tiles. The primary kernel softdtw global tiled called with the wavelength number of blocks and thread size of tile width.
- In this kernel, the row and column index for each tile is calculating and passing as a global variable to the corresponding kernel for the tiles' computation.
- The third step is the main kernel to process inter-tiles in parallel. Shared memory employed to keep the tile within the cache and improve the overall performance. As we mentioned earlier, we need to calculate the soft-min for the dependent cells. Therefore, the soft-min calculated for the up, left and diagonal upper left index.(equation 4)

$$D(i,j) = Soft - min \begin{cases} D(i-1,j) \\ D(i,j-1) \\ D(i-1,j-1) \end{cases} \tag{4}$$

Drawbacks: Due to the usage of global barriers, there would be lower utilization for the GPU. Fewer threads would be available at the beginning and at the end of each tiles process. Also, less tiles are accessible toward the beginning and end of wave iterations. Therefore, device SM remain idle during the initiation and end of the process.

Diagonal-major layout

Because the data dependency structure of the DTW algorithm results in elements of the distance and cost matrices on the same antidiagonal being processed in parallel, storing these matrices in row-major or column major order will cause a performance impact from cache misses and non-coalesced accesses to global memory.

If the data is first rearranged into an antidiagonal-major layout, then at each iteration of the wavefront loop, processor threads will make coalesced accesses to data elements that are laid out contiguously in memory. As illustrated in Figure 2, this transforms an mxn matrix that must be iterated over diagonally, into a $(m+n-1)x(min(m,n))$ matrix that can be iterated from top to bottom, with one thread assigned to each column. At each iteration step (i.e. each row of the diagonal-major matrix), contiguous array elements are read from memory into cache and written from cache back to memory, resulting in coalesced accesses and fewer cache misses.

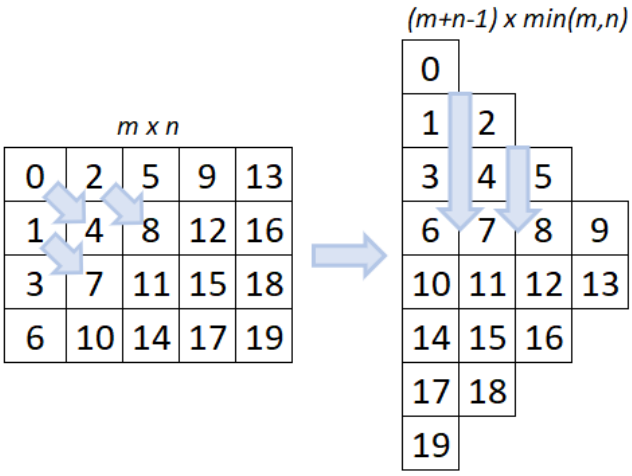

Figure 2: Cost matrix transformation to antidiagonal-major layout (arrows show iteration direction)

Shared memory stencil computation

As the program iterates diagonally across the distance matrix to find the optimal warping path, each cell in the path utilizes the previously computed results of three previous neighboring cells; if the iteration is visualized as proceeding from the upper left to the lower right corner of the matrix, the cost value in each cell depends on the (soft) minimmum of the costs in the cell above, the cell to the left, and the cell to the diagonal upper-left, which were computed in the previous two iterations

(Figure 3). If the cost matrix `R` resides in global memory, then non-contiguous accesses to `R[i-1][j]`, `R[i][j-1]` and `R[i-1][j-1]` will result in cache misses, incurring a significant performance cost. Since each element of `R` will be referenced up to three times in the computation of dependent cells, these cache misses can be avoided via a stencil computation using shared memory in CUDA. The stencil serves as a cache for the current and previous two diagonals; once a diagonal is no longer in use, its elements are written back to the cost matrix `R` in device global memory.

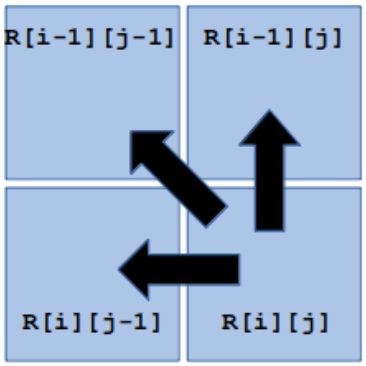

Figure 3: Data dependency direction between cells of the cost matrix. Each cell's cost computation depends on the costs of the three adjacent cells above, to the left, and to the upper-left.

The algorithm can be modified to use shared memory as follows:

```
D is a squared euclidean distance matrix of two time series
R is a cost matrix initialized to positive infinity except for R[0, 0] = 0
for each anti-diagonal of R starting from R[0, 0]
    if the current thread index < length of the current anti-diagonal
        copy R[i][j] from global memory into the stencil array
        read R[i-1][j], R[i][j-1] and R[i-1][j-1] from the stencil array
        compute cost as softmin(R[i-1][j], R[i][j-1], R[i-1][j-1]) + D[i-1][j-1]
        write the cost back to the stencil
        copy the cost in R[i-1][j-1] from the stencil back to global memory
```

Sakoe-Chiba bands

Sakoe-Chiba bands, proposed by Sakoe and Chiba in their 1978 paper "Dynamic programming algorithm optimization for spoken word recognition," introduce a "slope constraint" to the warping path to limit the maximum allowed warping distance beyond which a pair will not be considered in the optimal path calculation [11]. Pruning the search space by removing some of the extreme diagonals from consideration yields an approximation of the optimal warping path that can be calculated in sub-quadratic time.

This optimization is simple to implement for square matrices (i.e. DTW on time series of equal length) by checking that the absolute difference between the loop counter variables `i` and `j` does not exceed a fixed bandwidth threshold value (Figure 4). For rectangular matrices, since the leading diagonal does not end at the lower right corner, the implementation must be slightly modified to ensure that the counter variable along the longer of the two dimensions stays within a defined radius. Either way the result is a diagonal band matrix.

In a parallel programming environment such as CUDA, this optimization can also allow for the computation of the warping path using fewer threads, as threads assigned to cost matrix cells outside the band would go unused. Space savings can also be obtained if the bandwidth is known in advance, by storing the distance matrix and cost matrix in band storage format, omitting the zeroes in the corners.

While this technique produces only an approximation of the optimal path, in practice it has been shown to improve task performance by preventing pathological alignments where a very small portion of one time series maps onto a large portion of the other [5]. The width of the band can be a tunable hyperparameter for time series classification tasks.

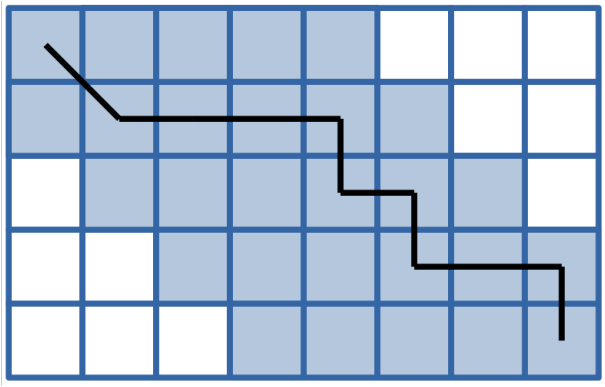

Figure 4: Illustration of one possible optimal DTW path for a length 5 series and a length 8 series with Sakoe-Chiba band radius = 1.

Test Data

For performance testing we selected datasets from the UCR Time Series Archive [12] and the UEA Multivariate Time Series Classification Archive [13].

Results

Discussion

Future Work

Our library provides CUDA kernels and C++ wrappers for computing pairwise Soft-DTW dissimilarity measures as well their gradients in parallel. Potential future work includes integrating this library with an optimization library that can iteratively minimize Soft-DTW cost to find barycenters among a set of time series, to assemble a nearest centroid classification system. Another area of potential is writing Python bindings to expose the Soft-DTW loss and gradient functions to deep learning frameworks such as TensorFlow or PyTorch, to enable the use of Soft-DTW loss as a training objective for recurrent neural networks. This will facilitate tasks such as classifying, predicting and generating sounds, gestures, and sensor data under the dynamic time warping geometry.

References

[1] M. Cuturi and M. Blondel, "Soft-DTW: A differentiable loss function for time-series," *arXiv:1703.01541 [stat]*, Feb. 20, 2018. arXiv: 1703.01541. [Online]. Available: http://arxiv.org/abs/1703.01541 (visited on 01/16/2021).

[2] D. Deriso and S. Boyd, "A general optimization framework for dynamic time warping," p. 23,

[3] E. J. Keogh and M. J. Pazzani, "Derivative dynamic time warping," in *Proceedings of the 2001 SIAM International Conference on Data Mining*, Society for Industrial and Applied Mathematics, Apr. 5, 2001, pp. 1–11, ISBN: 978-0-89871-495-1 978-1-61197-271-9. DOI: 10.1137/1.9781611972719.1. [Online]. Available: https://epubs.siam.org/doi/10.1137/1.9781611972719.1 (visited on 03/10/2021).

[4] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping," *ACM Transactions on Knowledge Discovery from Data*, vol. 7, no. 3, 10:1–10:31, Sep. 1, 2013, ISSN: 1556-4681. DOI: 10.1145/2500489. [Online]. Available: http://doi.org/10.1145/2500489 (visited on 03/07/2021).

[5] E. Keogh, "Exact indexing of dynamic time warping," in *Proceedings of the 28th international conference on Very Large Data Bases*, ser. VLDB '02, Hong Kong, China: VLDB Endowment, Aug. 20, 2002, pp. 406–417. (visited on 02/14/2021).

[6] D. Shen and M. Chi, "TC-DTW: Accelerating multivariate dynamic time warping through triangle inequality and point clustering," *arXiv:2101.07731 [cs]*, Jan. 19, 2021. arXiv: 2101.07731. [Online]. Available: http://arxiv.org/abs/2101.07731 (visited on 02/14/2021).

[7] L. Xiao, Y. Zheng, W. Tang, G. Yao, and L. Ruan, "Parallelizing dynamic time warping algorithm using prefix computations on GPU," in *2013 IEEE 10th International Conference on High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, Nov. 2013, pp. 294–299. DOI: 10.1109/HPCC.and.EUC.2013.50.

[8] H. Zhu, Z. Gu, H. Zhao, K. Chen, C. Li, and L. He, "Developing a pattern discovery method in time series data and its GPU acceleration," *Big Data Mining and Analytics*, vol. 1, no. 4, pp. 266–283, Dec. 2018, Conference Name: Big Data Mining and Analytics, ISSN: 2096-0654. DOI: 10.26599/BDMA.2018.9020021.

[9] M. Maghoumi, *Maghoumi/pytorch-softdtw-cuda*, original-date: 2020-05-02T23:28:24Z, Jan. 21, 2021. [Online]. Available: https://github.com/Maghoumi/pytorch-softdtw-cuda (visited on 01/23/2021).

[10] M. E. Belviranli, P. Deng, L. N. Bhuyan, R. Gupta, and Q. Zhu, "PeerWave: Exploiting wavefront parallelism on GPUs with peer-SM synchronization," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, Newport Beach California USA: ACM, Jun. 8, 2015, pp. 25–35, ISBN: 978-1-4503-3559-1. DOI: 10.1145/2751205.2751243. [Online]. Available: https://dl.acm.org/doi/10.1145/2751205.2751243 (visited on 03/03/2021).

[11] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, Feb. 1978, Conference Name: IEEE Transactions on Acoustics, Speech, and Signal Processing, ISSN: 0096-3518. DOI: 10.1109/TASSP.1978.1163055.

[12] H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. Keogh, "The UCR time series archive," *arXiv:1810.07758 [cs, stat]*, Sep. 8, 2019. arXiv: 1810.07758. [Online]. Available: http://arxiv.org/abs/1810.07758 (visited on 03/08/2021).

[13] A. Bagnall, H. A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, and E. Keogh, "The UEA multivariate time series classification archive, 2018," *arXiv:1811.00075 [cs, stat]*, Oct. 31,

2018. arXiv: 1811.00075. [Online]. Available: http://arxiv.org/abs/1811.00075 (visited on 03/08/2021).