Accelerating Dynamic Time Warping Subsequence Search with GPUs and FPGAs

Doruk Sart, Abdullah Mueen, Walid Najjar, Eamonn Keogh
University of California, Riverside
{sartd, mueen, najjar, eamonn}@cs.ucr.edu

Vit Niennattrakul
Department of Computer Engineering
Chulalongkorn University,Thailand
g49vnn@cp.eng.chula.ac.th

***Abstract*—Many time series data mining problems require subsequence similarity search as a subroutine. Dozens of similarity/distance measures have been proposed in the last decade and there is increasing evidence that Dynamic Time Warping (DTW) is the best measure across a wide range of domains. Given DTW's usefulness and ubiquity, there has been a large community-wide effort to mitigate its relative lethargy. Proposed speedup techniques include early abandoning strategies, lower-bound based pruning, indexing and embedding. In this work we argue that we are now close to exhausting all possible speedup from software, and that we must turn to hardware-based solutions. With this motivation, we investigate both GPU (Graphics Processing Unit) and FPGA (Field Programmable Gate Array) based acceleration of subsequence similarity search under the DTW measure. As we shall show, our novel algorithms allow GPUs to achieve two orders of magnitude speedup and FPGAs to produce four orders of magnitude speedup. We conduct detailed case studies on the classification of astronomical observations and demonstrate that our ideas allow us to tackle problems that would be untenable otherwise.**

Keywords- time series; similarity search; dynamic time warping; FPGA; GPU;

I. INTRODUCTION

Subsequence similarity search, the task of finding a region of much longer time series that matches a specified query time series within a given threshold, is a fundamental subroutine in many higher level data mining tasks such as motif discovery [15], anomaly detection [3], association discovery, and classification [16][1].

More than one hundred different distance measures for time series have been proposed in the last decade [9]; however there is increasing empirical evidence that Dynamic Time Warping (DTW) (which includes Euclidean Distance as a special case) is the best measure across a wide range of domains [5]. Given DTW's usefulness and ubiquity, there has been a large community-wide effort to mitigate its relative lethargy in the last decade. Speedup techniques for general search under DTW include various indexing methods [1][8][5]. Speedup techniques for the special cases of a subsequence similarity search that we consider here include early abandoning strategies, embedding and various "computation reuse" strategies [22]. A recent paper has shown that much of the *apparent* progress made in recent years is fatally flawed [17]. In particular, the speedup comes at the cost of allowing arbitrary false dismissals (we will expand on this surprising result in Section II.B).

Even if the apparent recent results had been correct, there still exist problems for which no current algorithms running on standard hardware can hope to solve in a reasonable amount of time. As a concrete example, entomologists need to examine telemetry gathered from insects for the occurrence of certain patterns known to be indicative of destructive (to host plants) behaviors. Entomologists at the University of California have created an archive of four hundred million data points of this data in the last four years, as part of an effort to understand and ultimately control just one insect, the Glassy-winged Sharpshooter (*Homalodisca vitripennis*). This insect causes tens of millions of dollars of damage to the grape industry. Searching this archive under the DTW distance for a single (relatively short) query pattern of length 360 takes nine days on a high-end desktop, using state-of-the-art algorithms. Similar stories can be told for astronomy (cf. Section VI.A), computational finance, motion capture processing data, industrial and medical domains.

After surveying and testing the current software solutions, and talking to several domain experts and practitioners, we have come to the conclusion that we are now close to exhausting all possible speedup from software approaches, and that we must turn to hardware-based solutions if we are to tackle the problems faced by real world practitioners. With this motivation, we investigate both GPU and FPGA based acceleration of subsequence similarity search under the DTW measure. The use of specialized hardware to allow subsequence similarity search requires a detailed understanding of both the hardware strengths and limitations, and of the DTW computation itself.

As we shall show, our novel algorithms allow GPUs, which are typically bundled with standard desktops and are thus essentially free, to achieve two orders of magnitude speedup. We show that if a domain practitioner is motivated enough to purchase an FPGA, which can cost as little as a few thousand dollars, our algorithm can achieve a speedup of four orders of magnitude.

It is important to note that we see our work as going beyond the claim that "*we have made an important algorithm faster*". A factor of say, two, speedup for an important algorithm is useful, but unlikely to make a significant difference to the community. However, a speedup factor of a thousand or more *really* has the potential to make a significant difference, because it allows problems to be tackled that are otherwise unimaginable.

II. DEFINITION AND BACKGROUND

For concreteness we begin with a formal definition of the problem and a discussion of why the current solutions are inadequate. We begin by defining the time series:

A *time series* T is a sequence of real numbers $t_1,t_2,\ldots,t_n$ representing n uniform samples of a measurement. A

1550-4786/10 $26.00 © 2010 IEEE
DOI 10.1109/ICDM.2010.21

subsequence $C_{s,m}$ of a time series T is the set of *m* samples starting at *s*. (i.e., $t_s,t_{s+1},...,t_{s+m-1}$, where $1 \le s \le n-m+1$).

Before we compare two time series under any distance measure, it is critical to normalize them to same mean and variance[1]. As noted in [9], "*without normalization time series similarity has essentially no meaning*".

The *z-normalization* of a time series T is defined as $\hat{T} = \hat{t}_1,\hat{t}_2,...,\hat{t}_n$, where $\hat{t}_i = (t_i - \mu) / \sigma$. Here, μ and σ are the sample mean and the sample standard deviation of T.

It is critical at this point to clarify a naive misunderstanding which is replete in the literature. If we are doing subsequence similarity search with our z-normalized query Q for the best matching subsequence in a much longer time series T, we *cannot* simply z-normalize T once and proceed. Instead, we must z-normalize every subsequence we extract from T. Note that in the case that T is not a batch dataset residing in its entirety in memory (or disk), but in a *data stream*, it would not even be logically possible to z-normalize it all, even if doing so gave meaningful results.

While DTW is defined to allow for the comparison of two time series of possibly different lengths, without losing the generality (see [20], Section 2), we will define it assuming time series of equal lengths.

Suppose we have two time series, $C = c_1,c_2,...,c_i,...,c_m$ and $Q = q_1,q_2,...,q_j,...,q_m$. The *Dynamic Time Warping* (DTW) distance between Q and C is denoted by $D(C,Q)$ and defined as below.

$$D(C,Q) = d(m,m)$$

$$d(i,j) = |c_i - q_j| + \min\{d(i-1,j), d(i,j-1), d(i-1,j-1)\}$$

$$d(0,0)=0;\ d(i,0)= d(0,j)=\infty; i=1,2,...,m;\ j=1,2,...,m$$

The *m*-by-*m* matrix, d, is called the *warping matrix*. In a warping matrix, each cell uses a value from either of the three previously computed neighbors. If we trace back the values used to compute the DTW (i.e. $d(m,m)$), we get the *warping path* describing the optimal alignment of T and Q .

The time complexity to compute the $D(C,Q)$ is $O(m^2)$, and the space complexity is also $O(m^2)$. If we only need the value of the distance (i.e. $d(m,m)$) we can delete the trace of the warping path, and thus, the space complexity can be reduced to $O(m)$ by storing only two columns of the matrix.

A. Definition of the Problem

Given a time series $T = t_1,t_2,...,t_n$ and a query $Q = q_1,q_2,...,q_m$, find the subsequence $C_{s,m}$ of T such that $D(\hat{C}_{s,m},\hat{Q})$, $1 \le s \le n-m+1$, is minimum.

Given the above definition, we could devise a brute force algorithm shown in Table 1, which takes $O(nm^2)$ time and $O(nm)$ space. For completeness, we also show the pseudocode for computing the DTW distance in Table 2.

[1] Some papers have suggested doing [0,1] or [-1,1] normalization instead. However, the authors do not seem to appreciate how brutally sensitive this method is to even small amounts of noise or a single outlier.

TABLE 1: SUBSEQUENCE SEARCH ALGORITHM

```
Procedure   SubsequenceSearch(T,Q)
            T: A time series of n points
            Q: Query time series of m points
1   z-Normalize(Q)
2   for s = 1 to n-m+1
3     z-Normalize(C_s,m)
4     Compute D(C_s,m,Q)
5     Update minimum if necessary
```

TABLE 2: DTW ALGORITHM

```
Procedure   D(C,Q)
            C: A time series of n points, C(0)= ∞
            Q: A time series of m points, Q(0)= ∞
1   s = 0
2   for i = 0 to m
3     d(i,s) = |C(1)-Q(i)|
4   s = s⊕1 // xor operation
5   for j = 2 to n
6     for i = 0 to m
7       d(i,s) = |C(j)-Q(i)| +
              min(d(i-1,s),d(i,s⊕1),d(i-1,s⊕1))
8     s = s⊕1
9   return d(n,s⊕1)
```

We have chosen the simplest possible problem definition with one query, one time series and the same subsequence length (*m*). There are more general subsequence search problems where many queries [24] and time series are involved, or where rotation/phase invariance is required under DTW [27][21]. However, all such problems can benefit directly from a speedup of the simple definition.

B. Why Current Software Solutions Are Not the Answer

As we hinted at above, the several *apparent* software solutions to the task at hand contain a serious error. We can best demonstrate this with a simple experiment.

Suppose we task a DTW subsequence search with the simple task of detecting the heartbeats of an individual, using one of that same individual's heartbeats.

We begin by downloading a long ECG sequence from a 61-year-old female and manually extracting a typical beat as our query [31]. We also manually extract some additional adjacent beats and compare them to our query, finding them to be an average distance of about 20.0, so we set our beat detector at a conservative threshold of 30.0. Figure 1 shows the beats detected in the first 1,800 datapoints, as we can see, the majority of the beats are missed. How could this be?

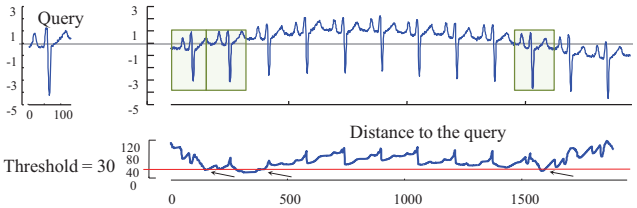

Figure 1: A query heartbeat *(left)* is scanned across an ECG trace. *(top-right)* Only three of the twelve beats are detected. Plotting the distance from the query to the relevant subsequence *(bottom)* reveals that slight differences in a subsequence's mean value (offset) completely dominate the DTW distance calculation, dwarfing any contribution from the similarity of the shape.

Note that while the local mean of the ECG trace starts at about zero, which is also the approximate mean of the query, the trace slowly rises to have a local mean of about 1.0, then descends below zero (allowing the detection of a single beat at about 1,500 as the mean crosses zero).

The problem is that the SPRING algorithm [22] does not (and, more critically *cannot*) normalize the offset or amplitude of the subsequences of the longer time series. It is therefore implicitly assuming that the query will happen to have the same offset and amplitude as the matching subsequence. However, in virtually every domain that assumption is unwarranted. For example, virtually all ECGs wander up and down as in our example, the effect is known as a *wandering baseline* [14]. Similar problems are observed in motion capture, astronomy, entomology, industrial process telemetry, EEGs, etc.

It is important to recognize that *there is no simple fix for this problem*. The SPRING [22] algorithm achieves its speedup by exploiting the redundancy of calculations in a sliding DTW matrix, but if each subsequence is z-normalized, as it must be to obtain meaningful results, then there will be no redundant calculations to exploit. For brevity we will conclude the discussion of the reasons why current software solutions do not work here. The interested reader can view [31], where we have many additional examples of the problem in real domains and a detailed discussion of why the obvious possible fixes for the problem we have just pointed out will not solve it. In our view, this problem is very unlikely to yield to a software solution that improves the time complexity of the brute force algorithm in Table 1.

III. ACCELARATION BY GPU

The GPU is a computing device that serves as a coprocessor for the CPU. It has its own device memory on the card and can execute many threads in parallel [18]. In this work we use the ubiquitous NVIDIA CUDA architecture, where multiple threads running on multiple processing cores execute the same program on separate data. This Single Instruction Multiple Data (SIMD) architecture allows us to map each normalization and DTW distance computation (Lines 3-4 in Table 1) to be executed in parallel on different segments of the time series.

Each CUDA function (i.e. kernel) is executed by an array of threads. Each of these threads is assigned an ID that it uses to determine memory addresses (i.e. the segment of the time series) it should operate on. The hardware is free to determine the mapping and scheduling of these threads on the available processing cores. A thread block is defined as a batch of threads that are guaranteed to run simultaneously and cooperate with each other through shared resources. The size of a thread block can be specified at runtime. The NVIDIA CUDA thread architecture can be found in [18].

The GPU implementation of Dynamic Time Warping consists of three main stages: (1) The CPU copies the values to the GPU memory. (2) The CPU calls the GPU kernel. (3) The CPU copies the output from the GPU.

In the first step, the CPU copies the whole time series (T in Table 1) to the global memory of the GPU. If the time series is larger than the available device memory, the CPU splits it into small batches and processes one batch at a time. This process introduces latency in the output but does not hamper the real-time processing, as the time to copy the data is in the range of milliseconds. Therefore, copying batches one at a time can tolerate a data arrival rate of hundreds of hertz without overflowing a buffer. Since the query is fixed for all of the batches, we copy it to the global memory in the beginning and keep it there throughout the execution.

In the second step, the CPU calls the kernel in the GPU. Every kernel thread operates on a specific sliding window in two steps: first, accessing the sliding window to compute the mean and variance, and second, computing the normalized DTW distance to the query. For both the steps, each kernel thread accesses a contiguous segment of m numbers from the time series T in the global memory. If we batch the threads responsible for successive sliding windows in a thread block, the memory accesses by these threads will result into coalesced accesses [7]. For example in Figure 2, a block of four threads is shown where the first memory accesses by these four threads require one read from the memory instead of four, because of the threads operating on contiguous locations in the memory.

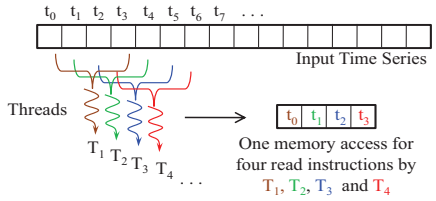

Figure 2: Division of work among threads. Memory accesses are coalesced by overlapping threads (for m=4).

In the second step, when the mean and variance are ready, each thread computes the DTW distance between the query and the normalized subsequence by Table 2. Note that in Table 2, the query Q is accessed $O(m^2)$ times whereas the subsequence C is accessed $O(m)$ times. It is important to maintain this distinction between Q and C although swapping Q and C in Table 2 would produce correct results. The reason is that Q is a fixed time series, and by the problem definition it is not changed during execution. Furthermore Q is much smaller than T so it can fit in the shared memory of the GPU device. Shared memory is a special memory locally available to the processor cores in a multi-processor of the GPU device. Shared memory is 150x faster than the global memory which is available to all of the cores in all of the multi-processors. Typically, shared memory is much smaller than the global memory and thus is the ideal place for the query time series.

In the beginning of the DTW computation, the kernel threads copy the query (Q) into shared memory from the global memory. The internal data structure for computing the DTW distance is two column vectors (d in Table 2) of size m. These vectors are best stored in the shared memory if they all fit together (e.g., in the case of very small m). Otherwise, they are stored in the global memory. With all of the variables in place, the DTW computation is performed. Each thread stores the computed distance in a global array indexed by the thread ID.

Finally in the third step, when all of the threads terminate, the CPU copies the array back to system memory. Although the algorithm looks for the *minimum* distance, it is much simpler and more efficient to copy back *all* of the distances to the CPU instead of computing the minimum in the GPU.

IV. Accelaration by FPGA

A. FPGA Implementation

The design of an FPGA configuration requires programming using a hardware description language (i.e., VHDL/Verilog). In this work, we use an open source C to VHDL compiler system, *ROCCC* [29], which allows us to describe the hardware in C language and generate the VHDL code automatically. *ROCCC* also optimizes the design mainly in three ways. First, it maximizes the throughput by exploiting loop and instruction level parallelism. Second, it reuses the data, and third, it generates a pipelined datapath to minimize the number of clock cycles [24].

Our FPGA design consists of two major blocks: *Normalizer* and *Warper*, to normalize the input data and run the actual DTW matrix calculations, respectively (Figure 3). Input data streamed into the system are first given to a First-In-First-Out (FIFO) buffer. The size and input ratio of this FIFO can be adjusted according to the FPGA interconnection mechanism. However, the output of the FIFO generates one sample (8 bits) every clock cycle. Next, the output of the FIFO is fed into the Normalizer module. Initially, Normalizer waits until the first window is received. Every following normalization operation reuses m-1 operands of the previous operation, where m is the query length. After the first output is produced, a new output is generated every clock cycle. This output is given to another FIFO, which acts as the intermediate memory component between the Normalizer and the Warper.

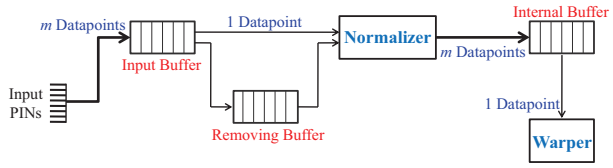

Figure 3: FPGA Block Diagram. Thick lines are for m-point wide connections. Thin lines are for one-point wide connections. Buffers are simple FIFOs.

Internally, a trivial Normalizer module stores m-partial sliding windows. In every clock cycle, it updates statistics for all of the partial windows and outputs the window for which the normalization is complete. Thus, it needs quadratic $O(m^2)$ space in the FPGA and does not scale with larger query lengths. In order to support larger query lengths, we implemented an *online Normalizer*, which does not remember intermediate states. It computes the mean (μ) and standard deviation (σ) online and normalizes exactly one window in every clock cycle. Thus, it needs linear $O(m)$ space in the FPGA. Although the trivial Normalizer has shown better performance in speed due to less overhead, it does not make any difference in the overall system performance. The reason for this is that the Warper module is the real performance bottle-neck as described later.

The Online Normalizer consists of three sub-units, as shown in Figure 4. The first unit calculates the sum and sum of squares of all the inputs in a sliding window fashion, by adding the new value while subtracting the oldest value to be removed from the sum. Initially, "to remove" value is zero until the first window is completely received through the "Datapoint" input. When switching to the next window, the very first value of the first window is given to "to remove" and the sum for the second window is obtained at the output. This output is also given to the Normalize Divider sub-unit, where the mean and the standard deviation of the latest window are obtained. The input stream is provided to the third unit through a buffer. The size of this buffer depends on the delay of the first two modules. The third module must wait until the corresponding mean and standard deviation values are available for a given window. This delay is provided by the Datapoint Buffer, which is automatically added by *ROCCC*. The unit then runs the actual normalization function. The generated normalized data is provided to the systolic array (warper) through a buffer, as shown in Figure 3.

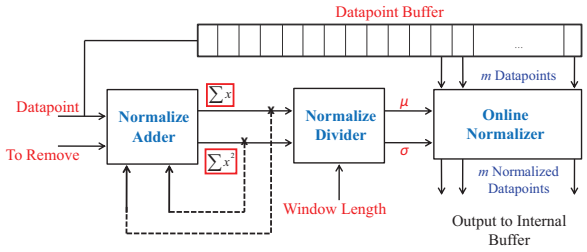

Figure 4: Online Normalization Unit. The sum and sum of squares are obtained in a sliding window approach, by adding the new input and subtracting the oldest value. The input datapoints are delayed through the Datapoint Buffer, to make sure that the correct mean and standard deviation are used.

The Warper module is implemented as a systolic array [2]. A systolic array consists of data processing units connected in a matrix fashion. These data processing units (i.e. cells) share the information with their neighbors immediately after processing. Using *ROCCC*'s built-in systolic array generator, we simply obtain the hardware description of the Warper module. Structurally, the Warper module is the same for any window size, except for the size of the systolic array. This size can be adjusted in the *ROCCC* code by tuning a parameter. A Warper module generates one DTW distance between the normalized sliding window and the query time series in every m clock cycle where m is the window size/query length. Since the normalization unit is m times faster than the Warper unit, we place multiple Warper units to operate on separate normalized windows generated by the normalization unit. Ideally, if we had unlimited FPGA area, we could place m Warper modules to get the maximum processing speed of one DTW distance in every clock cycle. When multiple Warper modules are in place, the Internal Buffer output is fed into them in a round robin fashion.

V. EVALUATION

In this section, we show the performances for the DTW subsequence search problem in different hardware settings. We use the following platforms: **Software:** Intel Xeon E5540 CPU at 2.53 GHz, **SSE:** Intel i7- 920 CPU at 2.66 GHz, **GPU:** NVIDIA Tesla C1060 with 240 cores and **FPGA:** Xilinx Virtex 5 LX-330.

The SSE (**S**treaming **S**ingle Instruction Multiple Data (SIMD) **E**xtensions) is an instruction set extension to Intel's x86-architecture. It makes use of 128-bit SSE registers and can merge four 32-bit data to operate concurrently. The software implementation proposed in Table 1 can be parallelized by making use of data independencies. We execute SSE instructions while normalizing every sample by the same μ and σ. The performance improvement is therefore not significant compared to the software-based solution.

In Figure 5, we show the time required to answer a query of length 128 by different hardware settings. We achieve the highest speedup over the software through FPGA acceleration, which is 4000 times faster in the best case scenario. GPU acceleration is 36.3 times faster, on average. All of the results reported here use 8-bit integers to represent the values in the time series.

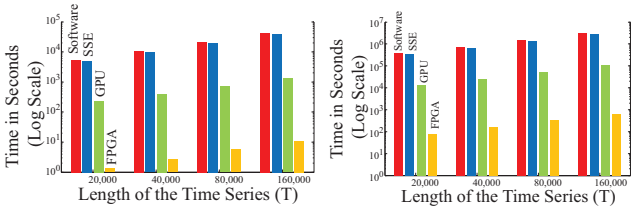

Figure 5: Comparison of execution times with different lengths of the input time series, in LOG scale. The query length is 128 (***left***) and 1024 (***right***). For GPU, block size is 512.

The FPGA performance results are obtained after placement and routing operations. We use one trivial Normalizer unit and eight Warper units. The Normalizer for window size 128 requires 13% of the target device and runs at 180MHz. Each Warper unit requires 7% of the area and run at 240MHz. The Normalization unit provides an output for each clock cycle. The Warper requires 128 clock cycles. Therefore, the Warper unit runs more slowly than the Normalizer. At 240 MHz, one window of length 128 can be processed through the Warper in 128/240M seconds. However, using 8 Warpers reduces this to 16/240M seconds. In other words, 15 million samples (windows) can be processed per second.

In Figure 5, we show the execution times for a fixed window size of 1024, the FPGA provides a maximum speedup of 4500x and the GPU achieves a speedup of 29x over software. For the window size 1024 on the FPGA, we use the online Normalizer, as shown in Figure 4. This module runs at a clock frequency of 180MHz and requires 83% of the FPGA logic. The Warper module runs at 250MHz and requires 9% of the logic. In this case, we can only place one Warper module safely. Although the area is dominated by the Normalizer, the throughput of the system is still determined by the Warper module. The Warper module requires 1024 clock cycles per cycle. One sample (window) can be processed in 1024/250M seconds. This results in a throughput of 244 thousand samples per second.

In Figure 6 we show the responses of different methods while varying the size of the query. Recall the methods have the same time complexity of $O(nm^2)$. The responses show a clear quadratic growth for software and SSE methods. Our hardware acceleration techniques are much slower in growth because of the parallelism our techniques achieve. The trends in the figure clearly show that our techniques will remain tenable for larger window size while the software methods are already intractable.

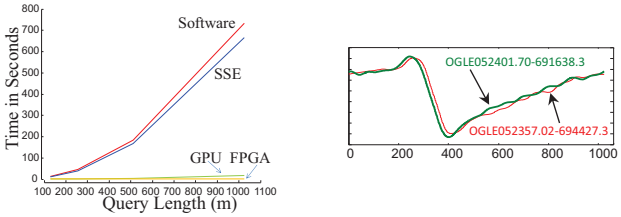

Figure 6: (***left***) Execution times for a fixed time series of length 40,000 with varying query sizes (m). (***right***) Two star light curves that happen to be very similar. If we happen to know the class label of one, we may predict that the other is in the same class.

VI. EXPERIMENTAL CASE STUDIES

A. *Case Study in Astronomy*

A star light curve is a graph which shows the brightness of a stellar object over a period of time. Figure 6 shows two examples. The reasons why the stars change their brightness include planetary transits, self-occluding binary systems, cataclysmic or explosive events (nova or supernova) and unknown reasons. The study of light curves has led to the discovery of pulsars, extra-solar planets, supernovae, the rate of expansion of the universe, etc. [19][21].

It is difficult to overstate how many star light curves exist. Looking backwards, there are over 500,000 glass photographic plates at Harvard University that were exposed between 1885 and 1993 [30]. These are currently being digitized to yield millions of light curves. Looking ahead, this year sees work starting on the Large Synoptic Survey Telescope (LSST), a wide-field "survey" reflecting telescope that will photograph the available sky every three nights. It is estimated that LSST will produce billions of light curves in the next decade.

As both old and new light curves come online, an obvious thing to do is to classify them [19]. Astronomers do have a large number of classified light curves; in some cases they can obtain what is effectively *ground truth* by obtaining extra features for relatively close stars.

While it is possible to extract a single light curve cycle, there is no well-defined starting point. Astronomers have an algorithm called *universal phasing* to produce a canonical alignment for light curves, but bemoan the fact that this is "*...an operation that scales poorly to massive data sets*". However, as we shall see, in addition to poorly scaling, the universal phasing algorithm does not work as well as astronomers believe.

We obtained a three-class star light curve dataset which had been universally phased by astronomers at (*blinded*). We

created a test set with just 128 objects, and a training set of 1024 objects. Each light curve was normalized to have a length of 1024 (as is the practice in astronomy).

We measured the accuracy of Euclidean distance and DTW, obtaining accuracies of 80.47% and 86.72%, respectively. This tells us that "warping" is useful in this domain, something that had been suspected before [27]. However, rather than stopping here, we decided to test the universal phasing assumption. Suppose we ignored it and tested DTW for all possible alignments/shifts. To our knowledge this has never been attempted before, presumably because the rotation invariance version of DTW (rDTW) is $O(n^3)$, which is untenable for a CPU. After testing the rotation-invariant versions of both Euclidean distance and DTW, we found that the accuracies jumped to 81.2% and 91.4%, respectively. Clearly, universal phasing does not produce perfect alignments.

TABLE 3: ACCURACIES AND TIMINGS FOR CLASSIFYING 128 LIGHT CURVES AGAINST A TRAINING SET OF 1,024, WITH ALL SEQUENCES OF LENGTH 1,024. VALUES IN PARENTHESES ARE ESTIMATED BASED ON SHORTER EXPERIMENTS.

	Accuracy	Time FPGA	Time GPU	Time CPU
ED	80.47%	<1.0 seconds	<1.0 seconds	2.5 seconds
rED	81.25%	<1.0 seconds	55.3 seconds	43.6 minutes
DTW	86.72%	<1.0 seconds	43.6 seconds	35.4 minutes
rDTW	91.41%	9.54 minutes	22.7 hours	(42 days)

VII. CONCLUSION[2]

We have shown subsequence similarity search is an important problem, but the current software based solutions cannot provide adequate speed to tackle many diverse domains. We have shown that hardware based solutions offer the necessary speedup. We have designed (*correctly normalizing*) DTW similarity search algorithm for GPUs and FPGAs and placed all code in the public domain [31].

REFERENCES

[1] Athitsos, V., Papapetrou, P., Potamias, M., Kollios, G. and Gunopulos, D. *Approximate Embedding-based Subsequence Matching of Time Series.* SIGMOD Conference 2008: 365-378.

[2] Buyukkurt B. and Najjar W. *Compiler Generated Systolic Arrays For Wavefront Algorithm Acceleration on FPGAs*. International Conference on Field Programmable Logic and Applications, 2008: 655-658.

[3] Chandola, V., Cheboli, D. and Kumar, V. *Detecting Anomalies in a Time series Database.* CS Technical Report 09-004, January 2009, Computer Science Department, University of Minnesota.

[4] Chu, S., Narayanan, S. and Jay Kuo, C.-C. *Efficient Rotation Invariant Retrieval of Shapes using Dynamic Time Warping with Applications in Medical Databases.* In IEEE International Symposium on Computer-Based Medical Systems (CBMS), Special Track on Data Mining, 2006.

[2] Dr. Najjar's work was funded by NSF CCF0905509 and CCF0811416. Dr. Keogh's work was funded by NSF 0803410 and 808770.

[5] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X. and Keogh, E. *Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures.* VLDB 2008.

[6] Frost, K. E. and Groves, R. L. *Seasonal Infectivity of Aster Leafhoppers in Carrot*. Technical Report Department of Entomology, University of Wisconsin-Madison. www.entomology.wisc.edu/vegento/xtras/proc/2009_asterLeafhoppers.pdf

[7] He, B., Yang, K., Fang, R., Lu, M., Govindaraju, N. K., Luo, Q. and Sanderv P. *Relational Joins on Graphics Processors* SIGMOD 2008.

[8] Keogh, E. (2002): Exact Indexing of Dynamic Time Warping. VLDB 2002: 406-417

[9] Keogh, E. J. and Kasetty, S. *On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration.* Data Min. Knowl. Discov. 7(4): 349-371 (2003).

[10] Keogh, E., Xi, X., Wei, L. and Ratanamahatana, C. A. (2006). *The UCR Time Series Classification/Clustering Homepage*: www.cs.ucr.edu/~eamonn/time_series_data/

[11] Kindt, F., Joosten, N. N. and Tjallingii, W. F. *Electrical Penetration Graphs of Thrips Revised: Combining DC- and AC-EPG Signals.* Journal of Insect Physiology 52: 1–10.

[12] Lienhart, G., Kugel A., Männer R. *Using floating-point arithmetic on FPGAs to accelerate scientific N-body simulations*. FCCM 2002.

[13] McLean, D. L. and Kinsey, M. D. *A Technique for Electronically Recording Aphid Feeding and Salivation*, Nature 202 (1964), pp. 1358–1359.

[14] Mneimneh, M. A., Yaz, E. E., Johnson, M. T. and Povinelli, R. J. *An Adaptive Kalman Filter for Removing Baseline Wandering in ECG Signal*. Computers in Cardiology, vol. 33, pp.253−256, 2006.

[15] Mueen, A., Keogh, E. J. and Bigdely-Shamlo, N. *Finding Time Series Motifs in Disk-Resident Data,* ICDM 2009: 367-376.

[16] Nam, H., Lee, K. and Lee, D. *Identification of Temporal Association Rules from Time-Series Microarray Data Sets.* BMC Bioinformatics, vol. 10 (Suppl 3):S6, March 2009.

[17] Niennattrakul, V. and Ratanamahatana, C. A. *Meaningful Subsequence Matching under Time Warping Distance for Data Stream.* PAKDD 2009: 1013-1020.

[18] NVIDIA CUDA Programming Guide. Version 2.3. http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf

[19] Protopapas, P., Giammarco, J. M., Faccioli, L., Struble, M. F., Dave, R. and Alcock, C. *Finding Outlier Light Curves in Catalogues of Periodic Variable Stars*. Mon. Not. R. Astron. Soc. 369(2), 677–696 (2006).

[20] Ratanamahatana, C. and Keogh, E. J. *Three Myths about Dynamic Time Warping Data Mining.* SDM 2005.

[21] Rebbapragada, U., Protopapas, P., Brodley, C. E. and Alcock, C. *Finding Anomalous Periodic Time Series: An Application to Catalogs of Periodic Variable Stars*. Machine Learning, 74(3), p. 281, 2009.

[22] Sakurai, Y., Faloutsos, C. and Yamamuro, M. *Stream Monitoring under the Time Warping Distance.* ICDE 2007: 1046-1055.

[23] Salzberg, S. L. *On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach.* Data Mining and Knowledge Discovery, 1(3), 1997.

[24] Villarreal, J, Park, A., Najjar, W. and Halstead, R. *Designing Modular Hardware Accelerators in C With ROCCC 2.0*, in The 18th An. Int. IEEE Symp. On Field-Programmable Custom Computing Machines (FCCM), Charlotte, NC, May 2010.

[25] Wei, L., Keogh, E. J., Van Herle, H. and Mafra-Neto, A. *Atomic Wedgie: Efficient Query Filtering for Streaming Times Series.* ICDM 2005: 490-497.

[26] Wilson, D. R. and Martinez, T. R. (1997). *Instance Pruning Techniques.* ICML'97, Morgan Kaufmann, pp. 403-411.

[27] Yankov, D., Keogh, E. J., Wei, L., Xi, X. and Hodges, W. L. *Fast Best-Match Shape Searching in Rotation-Invariant Metric Spaces.* IEEE Transactions on Multimedia 10(2): 230-239 (2008).

[28] Zhu, Y. and Shasha, D. *Warping Indexes with Envelope Transforms for Query by Humming*. SIGMOD Conference 2003: 181-192

[29] http://roccc.cs.ucr.edu

[30] http://hea-www.harvard.edu/DASCH/index.php

[31] Supporting webpage: For code and data. http://www.cs.ucr.edu/~mueen/GPU_DTW/index.html