Accelerating the Dynamic Time Warping Distance Measure using Logarithmetic Arithmetic

Joseph Tarango, Eamonn Keogh, Philip Brisk
University of California, Riverside
Department of Computer Science and Engineering
Riverside, CA, USA
{jtarango, eamonn, philip}@cs.ucr.edu

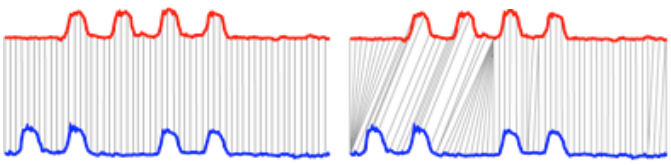
***Abstract*—This paper describes an application-specific embedded processor with instruction set extensions (ISEs) for the Dynamic Time Warping (DTW) distance measure, which is widely used in time series similarity search. The ISEs in this paper are implemented using a form of logarithmic arithmetic that offers significant performance and power/energy advantages compared to more traditional floating-point operations.**

Keywords—Time series; similarity search; application-specific processor; Instruction Set extension (ISE); Euclidean Distance (ED); Dynamic Time Warping (DTW); floating-point arithmetic; logarithmic arithmetic

I. Introduction

Subsequence similarity search (also known as *time series similarity search*) is the task of finding a region within a much longer time series that matches a specified query time series within a given threshold. It is useful in its own right as an exploratory tool, and it is the core subroutine in many higher-level data mining tasks such as motif discovery, anomaly detection, association discovery, and classification.

One simple and straightforward distance measure for time series similarity is Euclidean Distance (Fig. 1a), which considers a one-to-one mapping in time between the points on the two time series. While more than one hundred different distance measures for time series similarity have been proposed for this task, there is increasing empirical evidence that Dynamic Time Warping (DTW, Fig. 1b), which includes ED as a special case, is the best measure across a wide range of domains, from robotics to medicine [9]. Unlike ED, DTW can find similarities between time series that have relative minor offers in time. Given DTW's usefulness and ubiquity, there has been a large community-wide effort to mitigate its relative lethargy in the last decade.



(a) (b)

Fig. 1. Euclidean Distance (a) and Dynamic Time Warping (DTW) (b) are distance measures for time series similarity search. DTW enables realignment in time, while Euclidean Distance does not.

DTW is expected to play a central role in the emerging Internet of Things (IoT), in which low-cost sensor-based devices connected to the Internet produce time-series data. A representative device could be a small wearable computer that performs real-time physiological monitoring. It must be able to generate and analyze time-series data in real-time, while minimizing energy consumption to maximize battery lifetime.

IoT devices are likely to be realized using microcontroller-based SoCs in order to minimize cost and power. Such an SoC could benefit from application-specific customization for time series similarity search, due to its overall ubiquity; however, full-blown ASIC accelerators would likely be too large and too costly. A cheaper solution is to customize the microcontroller at the core of the SoC with a small set of application-specific instruction set extensions (ISEs) that offer much of the desired performance and/power energy benefits that one would obtain from an ASIC, but at a fraction of the cost [7].

This paper explores the use of logarithmic arithmetic, rather than floating-point, in a set of four ISEs that have already been shown to be beneficial for DTW. At the highest compiler optimization level (gcc -O3), the logarithmic ISEs improved performance and energy consumption by identical factors of 1.8x compared to traditional floating-point ISEs.

II. Related Work

DTW was introduced in 1968 for speech processing [8]; since then, it has been found useful for time series similarity search in many application domains. The DTW measure can be computed in quadratic time using dynamic programming. In the context of an IoT device, DTW is invoked each time a signal is sampled for each query that the sample time series is being matched against. The overhead is prohibitive for use in a power- and resource-constrained context such as IoT. Consequently, there has been a wealth of work that tries to perform similarity search in a way that avoids computing DTW when more efficient analyses can prove that the current time series subsequence and query exceed the threshold for a match.

Proposed speedup techniques include early abandoning strategies, lower-bound based pruning, indexing and embedding into a lower dimensional metric space [9]; however recent work has argued that we are now close to exhausting all possible speedup from software, and that we must turn to hardware-based solutions [5].

Ref. [5] argues that as DTW subsequence search is carefully optimized, an increasingly large fraction of the time is spent in Z-normalizing each subsequence before it is passed to the DTW function. Z-Normalization ensures that the query and the subsequence have the same mean and standard deviation, and are thus commensurate. Ref. [5] demonstrates that is possible to do the Z-Normalization in an online fashion, and moreover, that it is possible to interleave the early abandoning calculations with the online Z-normalization. In other words, DTW (or a lower bound of DTW used for early abandoning) can be computed incrementally while Z-normalizing the same data point. Thus, early abandoning prunes not just the distance calculation step, but subsequent normalization steps.

Ref. [5] demonstrated notable performance improvements on a desktop PC. The code was written in C and used double-precision floating-point arithmetic. In the embedded domain, Ref. [7] ported the software to an embedded processor and introduced four floating-point ISEs for acceleration; the system was prototyped and evaluated using an FPGA. At the highest compiler optimization level (gcc -O3), the floating-point ISEs improved performance by a factor of 1.4x and energy consumption by a factor of 2.9x compared a baseline variant of the processor which included a floating-point ALU.

The contribution of this work can be best viewed as an extension to Ref. [7]. In Ref. [7], the ISEs were implemented using compound floating-point operators that were optimized for an FPGA implementation by the FloPoCo arithmetic compiler [2]. This work replaces the floating-point operations within each ISE with operations based on logarithmetic arithmetic; the result is further improvements in performance and energy consumption by identical factors of 1.8x.

III. Floating-Point ISEs for Time Series Similarity Search based on DTW

Table I lists the four ISEs reported in Ref. [7]: **ISE-Norm**, **ISE-DTW**, **ISE-Accum**, and **ISE-SD**. In the discussion that follows the two time series whose similarity is being measured are the query Q and the candidate C; q_i and c_i are the i^{th} respective data points of the query and candidate.

ISE-Norm is normalization each candidate data point c_i; c_i' is the normalized data point. The mean μ_C and standard deviation σ_C of the candidate data point values are computed in an online fashion as described in Ref. [5, Section 4.2.1].

ISE-DTW is the recurrence relation at the core of the dynamic programming algorithm that computes DTW. The recurrence relation computes $\gamma(i,j)$, which is the minimum cost warping path between sub-query $Q_{1..i} = \{q_1, ..., q_i\}$ and sub-candidate $C_{1..j} = \{c_1, ..., c_j\}$, and $d(q_i, c_j) = |q_i - c_j|$ is the distance between points q_i and c_j.

TABLE I. The Four Floating-point ISEs Identified in Ref. [7].

ISE-Norm	$c_i' = \frac{c_i - \mu_C}{\sigma_C}$
ISE-DTW	$\gamma(i,j) = (q_i - c_j)^2 + min\{\gamma(i-1,j-1), \gamma(i-1,j), \gamma(i,j-1)\}$
ISE-Accum	$a = a + b$
ISE-SD	$z_i = (q_i - c_i)^2$

ISE-Accum is a floating-point accumulation operation that is called frequently by the LB_{Keogh} [4] lower bound; as described in Ref. [5], computation of LB_{Keogh} can be interleaved with online Z-normalization for better performance.

ISE-SD: The squared difference of two points is part of the computation for the Euclidean Distance measure. LB_{Keogh} computes the Euclidean Distance between the candidate C and the closer among a pair of auxiliary sequences $\{U, L\}$, which form upper and lower bounds on the query Q.

IV. Floating-Point and Logarithmic Arithmetic

The floating-point and logarithmic number formats used in the ISEs described in this paper are based on representations used in the FloPoCo compiler [2]; the floating-point operations and ISEs are not compliant with the IEEE-754 standard.

A. Floating-Point Number Format

A floating-point number P is a tuple $P = (Z, s, E, M)$, where Z is a two-bit exception field, s is the sign bit, E is the exponent, and M is the mantissa; let $e = |E|$ and $m = |M|$ be the number of bits in the exponent and mantissa respectively. The exception values are *00* for zero, *01* for normal numbers, *10* for infinity, and *11* for not-a-number (NaN).

A normalized mantissa has a hidden '1' that is not explicitly encoded, so the actual value is $1.M$, where the m mantissa bits represent the fractional part of the number. The range of a normalized mantissa is

$$1 \leq 1.M \leq 2 - 2^{-m}. \quad (1)$$

For sake of brevity, we do not discuss denormalized mantissas.

The exponent E is base-2 and a bias of $B = 2^{e-1} - 1$. When $Z = 01$, the value p represented by a floating-point number P is

$$p = (-1)^s (1.M) \times 2^{E-(B-1)}. \quad (2)$$

B. Logarithmic Arithmetic

The key idea behind logarithmic number systems is to represent a positive value X by its logarithm, $logX$. The following three identifies explain several key advantages to using logarithmic representations:

$$log(XY) = logX + logY, \quad (3)$$

$$\log\left(\frac{X}{Y}\right) = logX - logY, \text{ and} \quad (4)$$

$$\log(X^a) = alogX. \quad (5)$$

In other words, multiplication, division, and exponentiation in a logarithmic number system can be implemented using fixed-point addition, subtraction, and multiplication respectively.

Addition and subtraction in a logarithmic number system are somewhat more complicated:

$$log(X \pm Y) = logX + log(1 \pm 2^{logY - logX}). \quad (6)$$

The right hand-term is typically implemented by multipartite table lookup and addition [3]; optimizing hardware designs for this term has been an active resarch area since the 1970s [6].

C. Logarithmic Number Format

A logarithmic number L is a tuple $L = (Z, s, I, F)$, where Z is a two-bit exception field (identical to the floating-point format), s is the sign bit, and I and F are the integral and fractional parts of the value of $logX$, which is represented as a two's complement fixed-point number. When $Z = 01$, the value l represented by a floating-point number L is

$$l = (-1)^s(I.F) = (-1)^s logX. \quad (7)$$

D. Handling Error Accumulation

The online Z-normalization approach described in Ref. [5] is susceptible to the cumulative accumulation of floating-point error. Ref. [5] proposes to flush the accumulated error each time one million new time series data points are processed. To flush, the online Z-normalization is abandoned for the current candidate subsequence, and an offline (exact) Z-normalization is computed instead. When subsequent data points arrive, the online Z-normalization can then be resumed.

Our approach is somewhat different, and requires a detailed accounting of the bits in both the floating-point and logarithmic number formats. In the floating-point number format, $e = 11$ and $f = 52$; in the logarithmic number format, there are 11 integral bits and 52 fractional bits. With the sign and exception bits, there are 66 bits in total. In our implementation, the exception bits are computed internally by the ISEs, along with a 67^{th} valid bit, that is used for energy management. Pipeline stages that are not in use are shut down to conserve dynamic energy, and the valid bit is turned off; otherwise, it remains on. The valid bit serves as an extra guarantee that the staging, pipeline, and finishing registers always hold correct values.

If the valid bit is not set, the data produced by the ISE can be ignored. If it is set, a flush is triggered if the exception bits are infinity or NaN. If no flush is triggered, the result is transmitted back to the processor.

V. Logarithmic ISEs for DTW

The four ISEs listed in Table I were redesigned in the following way. Each data element enters and leaves each ISE in double-precision IEEE-754 floating-point format. Internally, the data element is converted to the logarithmic format from Section IV.C, the ISE operations are then performed using logarithmic operations, and the result is then converted back to the IEEE-754 compliant format. This ensures that all floating-point operations performed outside of the ISEs are IEEE-754 compliant and completely portable across processors.

The conversions between the double-precision IEEE-754 floating-point and logarithmic format in both directions were implemented using multipartite table lookup and addition [3]. Addition and subtraction in the logarithmic format were implemented as described in Ref. [1]. Multiplication (squaring for **ISE-SD**) and division (**ISE-Norm**) are implemented using 64-bit fixed-point adders and subtractors respectively.

ISE-DTW, which includes a 3-way minimum operation, is more complex. The three input values $\gamma(i-1, j-1)$, $\gamma(i-1, j)$, and $\gamma(i, j-1)$ are converted from the double-precision IEEE-754 floating-point format to the FloPoCo floating-point format described in Section IV.A; the minimum of the three values is computed in this format, and then converted to the logarithmic format. The two other input parameters q_i and c_j are converted directly to the logarithmic format; $(q_i - c_j)^2$ is then computed and added to the minimum value in the logarithmic format. The result is then converted back to the double-precision IEEE-754 floating-point format and returned to the processor.

For all ISEs, the 64-bit result that is sent back to the processor via the 64-bit FIFO interface is always in double-precision IEEE-754 floating-point format. A post-processing stage at the end of each ISE examines the two except bits and converts the value to NaN (as defined by the IEEE-754 format) if an exception occurs. After invoking each ISE, the software checks the result to see if it is NaN. If so, a flush is triggered and the query is immediately re-normalized offline. Online normalization resumes when the next data point is sampled.

VI. Experimental Setup

We prototyped the processor on a Xilinx EK-V6-ML605-G Virtex 6 development board using a MicroBlaze soft processor, which supports ISEs. The development board was connected to a desktop PC running Windows 7 x64. We compiled the DTW application using gcc 4.1.2, and synthesized all custom hardware using Xilinx Embedded System Design Software version 12.4; power and energy estimates were obtained using Xilinx XPower Analyzer [10].

The MicroBlaze runs at 100MHz and uses the PLB peripheral bus. Table II lists the instruction and data cache configurations that we used, which is the same as Ref. [7]. We enabled 64-bit fixed-point multiplication, a hardware divider, and a branch target cache with 2048 entries.

Four processor configurations are used in our experiments. The **Base** processor configuration is described above. Lacking a floating-point unit (FPU), all floating-point operations are implemented in software. **Base-FPU** includes a pipelined FPU which is accessed through ISEs for all standard floating-point operations. **DTW-FP** includes the four ISEs, implemented using double-precision floating-point arithmetic. It does not include an FPU, so all arithmetic operations that operate on floating-point data elements and are not included in an ISE are implemented in software. **DTW-LOG** includes the same four ISEs implemented using logarithmic arithmetic as described in Section V. It does not include an FPU. Arithmetic operations that are not included in the four ISEs are implemented in software using a double-precision IEEE-754 compliant library.

The query length was set to 128. The input data set had one million double-precision floating-point time-series data values, which exceeded the FPGA's block RAM capacity. We modified the linker script to map data sections to the 512MB external DDR3 memory on the development board. As part of the configuration process, a PLB peripheral populates the DDR3 memory prior to executing the application.

TABLE II. MICROBLAZE INSTRUCTION AND DATA CACHE CONFIGURATIONS.

Parameter/Policy	I-Cache	D-Cache
Capacity	64KB	64 KB
Cache Line Length	32 bytes	32 bytes
Allocation Policy	Read/Write Allocate	Read/Write Allocate
Associativity	Direct-mapped	Direct-mapped
Victim Buffer Size	8 Victims	8 Victims
Other	1 Stream	Write-back

We selected the point-to-point Fast Simple Link (FSL) to provide a communication interface between the MicroBlaze register file and ISEs implemented as hardware accelerators. The FSL uses a 32-bit Master/Slave interface with optional FIFO data buffers; thus, each data element was separated into two 32-bit words. We created a set of Finite State Machines (FSMs) to transfer different numbers of data elements to/from the processor's register file, based on the needs of each ISE. Fig. 2 depicts the ISE interface. The latency of executing one ISE includes its pipeline depth plus the number of FSM cycles required for data transfers.

We introduced parameterized function calls into the source code to invoke calls to the FPU (for **Base-FPU**) and ISEs (for **DTW-FP** and **DTW-LOG**). Inline assembly separated data into 32-bit words and populated the FSL. For each invocation, the MicroBlaze was configured to execute a sequence of NOPs equal in length to the pipeline depth of the operator or ISE. At optimization level -O3 gcc unrolled loops and applied software pipelining to best utilize the FPU and FP ISEs; under the logarithmic number system, **ISE-DTW** required two cycles (ignoring data transfer overhead) while the other three ISEs required one cycle.

Our experimental setup is aligned as closely as possible to the prior work described in Ref. [1], which introduced the DTW-FP processor, which accelerates similarity search using DTW with using double-precision floating-point ISEs.

VII. EXPERIMENTAL RESULTS

A. FP and LNS ISEs

Figs. 3 and 4 report the latency and area of the FPU and all four ISEs implemented using FP and logarithmic arithmetic. The logarithmic ISEs have far lower latencies than the floating-point ISEs, and are significantly smaller as well. Floating-point operations incur extra overhead due to mantissa alignment (shifting), normalization (shifting), and rounding; the overhead of the the logarithmic operators is much smaller in comparison.

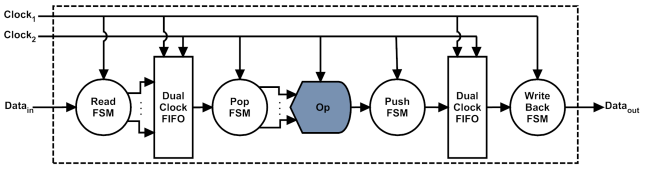


Fig. 2. ISE interface with dual-clock FIFOs and finite state machine (FSM) control [7, Fig. 15].

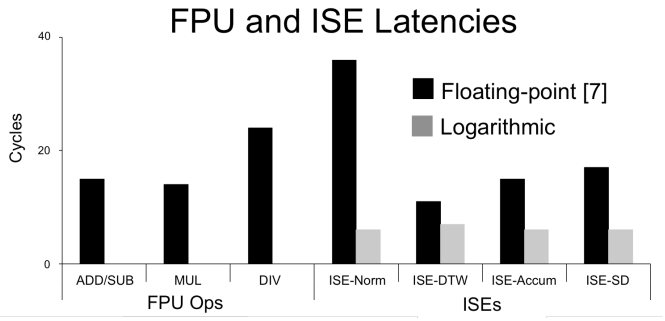


Fig. 3. Latencies of FPU operations (for the **Base-FPU**) processor configuration and the floating-point and logarithmic ISEs (for the **DTW-FP** and **DTW-LOG** processor configurations).

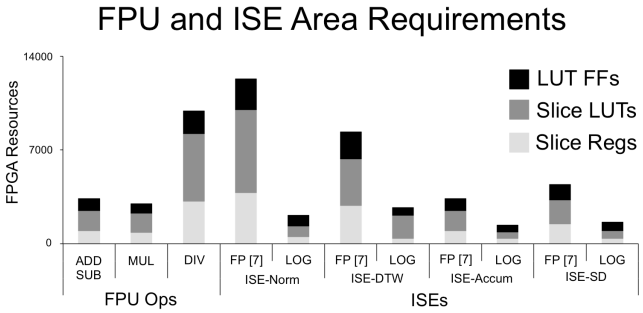


Fig. 4. Area (FPGA resource usage) for the FPU operations and the floating-point and logarithmic ISEs.

Fig. 5 shows that **DTW-LOG** outperforms **DTW-FP** significantly. At optimization level -O0, **DTW-LOG** with one ISE outperforms **DTW-FP** with four ISEs; at optimization levels -O1 and -OS, **DTW-LOG** with one ISE achieves equal performance to **DTW-FP** with four ISEs; and at optimization level -O3, **DTW-LOC** with two ISEs achieves equal performance to **DTW-FP** with four ISEs. In all cases, further increasing the number of available ISEs to **DTW-LOG** yields additional gains in performance.

The best overall performance is obtained by **DTW-LOG** with four ISEs at optimization level -O3; the speedup over **DTW-FP** with four ISEs at the same optimization level is 1.8x. The primary cause of this disparity is the overhead that floating-point operations incur due to mantissa alignment, normalization, and rounding.

An important detail to note is that flushing and offline normalization occurs much more frequently for **DTW-LOG** than for **DTW-FP**. The reason is that the logarithmic ISEs (including conversion to and from the double-precision IEEE-754 floating-point format) introduce more error than the floating-point ISEs. Since floating-point and logarithmic number systems have comparable dynamic ranges, this phenomenon is not *inherent* to the latter; it is simply a problem that arises due to tradeoffs made when creating arithmetic operators for the logarithmic number system. By using (exponentially) larger and more accurate lookup tables, much of this error accumulation could be suppressed. We have no investigated this issue further because our low-area implementation (Fig. 4) still manages to achieve good performance (Fig. 5).

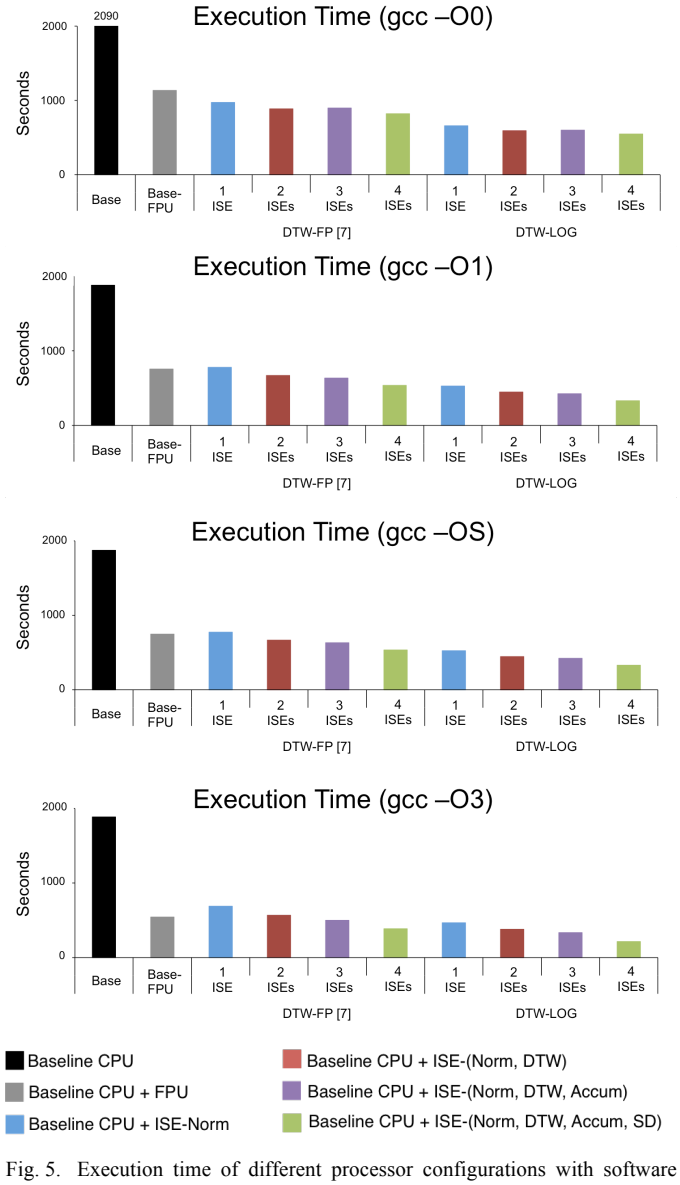


Fig. 5. Execution time of different processor configurations with software compiled using gcc optimization levels -O0, -O1, -OS, and -O3.

Fig. 6 reports the energy and peak power consumption of different processor configurations at the gcc -O3 optimization level. **Base** has the lowest peak power consumption because it does not have an FPU or any ISEs which consume dynamic power; however, it has the highest overall energy consumption because of its long execution time. The other processor configurations finish faster and can therefore shut down earlier, thus reducing total energy consumption.

Base-FPU, **DTW-FP**, and **DTW-LOG** consume far less energy than **Base** because they finish their executions much faster. Although **Base-FPU** has the lowest peak power consumption, **DTW-FP** and **DTW-LOG** consume less energy due to their performance advantages. Similarly, **DTW-LOG** consumes less energy than **DTW-FP** due to its performance advantage, as its advantage in terms of peak power (0.04 W) is negligible.

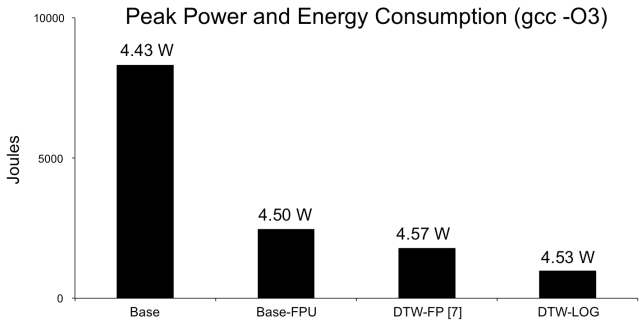


Fig. 6. Peak power and energy consumption of different processor configurations (DTW-FP and DTW-LOG use all four ISEs) with software compiled using gcc optimization level -O3.

VIII. Conclusion

This paper has shown that hardware accelerators for the dynamic time warping distance measure for time series similarity search perform better and consume less area and energy under logarithmic number systems than floating-point number systems. Several application-specific processors were prototyped on an FPGA, and hardware acceleration was realized through application-specific instruction set extensions. As software optimization alone seems unlikely to yield further improvements to this widely applicable algorithm, the next step is to explore further avenues for hardware acceleration include parallel implementations and executing the entire algorithm, not just ISEs, using a logarithmic number format.

Acknowledgment

This work was supported in part by NSF Grants CNS-1035603 and IIS-1161997. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the NSF.

References

[1] J. Detrey, F. de Dinechin: A Tool for Unbiased Comparison between Logarithmic and Floating-point Arithmetic. VLSI Signal Processing 49(1): 161-175 (2007)

[2] F. de Dinechin, B. Pasca: Designing Custom Arithmetic Data Paths with FloPoCo. IEEE Design & Test of Computers 28(4): 18-27 (2011)

[3] F. de Dinechin, A. Tisserand: Multipartite Table Methods. IEEE Trans. Computers 54(3): 319-330 (2005)

[4] E. J. Keogh, C. (A.) Ratanamahatana: Exact indexing of dynamic time warping. Knowl. Inf. Syst. 7(3): 358-386 (2005)

[5] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, E. J. Keogh: Addressing Big Data Time Series: Mining Trillions of Time Series Subsequences Under Dynamic Time Warping. TKDD 7(3): 10 (2013)

[6] E. E. Swartzlander Jr., A. G. Alexopoulos: The Sign/Logarithm Number System. IEEE Trans. Computers 24(12): 1238-1242 (1975)

[7] J. Tarango, E. Keogh, P. Brisk: Instruction set extensions for Dynamic Time Warping. CODES+ISSS 2013: 1-10

[8] T. K. Vintsyuk: Speech discrimination by dynamic programming". Kibernetika, 4(1): 81–88 (1968)

[9] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, E. J. Keogh: Experimental comparison of representation methods and distance measures for time series data. Data Min. Knowl. Discov. 26(2): 275-309 (2013)

[10] Xilinx, "Xilinx Power Tools Tutorial UG733 (v14.2)," July 25 2012.