UNIVERSITY OF CALIFORNIA
RIVERSIDE

Breaking Computational Barriers to Perform Time Series Pattern Mining at Scale and at the Edge

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Zachary Pierce Zimmerman

December 2019

Dissertation Committee:

Dr. Philip Brisk, Co-Chairperson
Dr. Eamonn Keogh, Co-Chairperson
Dr. Samet Oymak
Dr. Evangelos Papalexakis

The Dissertation of Zachary Pierce Zimmerman is approved:

Committee Co-Chairperson

Committee Co-Chairperson

University of California, Riverside

Acknowledgments

I would like to thank my advisors Dr. Philip Brisk and Dr. Eamonn Keogh, for supporting me through the PhD program, through all its ups and downs.

I would like to thank my colleagues Dr. Yan Zhu and Dr. Chin-Chia Michael Yeh for working with me in the earlier years of my PhD and letting me incorporate their earlier work on the matrix profile into my own.

I would like to thank my committee members, Dr. Samet Oymak and Dr. Vagelis Papalexakis; as well as my labmates Brian Crites, Jeffery McDaniel, Jason Ott, Amin Kalantar, Kaveh Kamgar, Shaghayegh Gharghabi, and Shmia Imani for always letting me bounce ideas off of them.

I would like to thank Dr. Nader Shakibay Senobari, for being both a coauthor and a friend through most of my time in the PhD program.

I would like to thank my friends (you know who you are) for being there with me through my education.

I would like to thank my aunts, uncles, cousins, other family members, and friends who have supported me at one point or another during my education. I don't know where I would be without your help.

I would like to thank my parents, for raising me to be inquisitive and curious about everything.

A number of previous publications were used in the preparation of this dissertation, and parts of them are reprinted with the following permissions.

To my friends and family,

who provided the foundations I stand on,

and the walls I lean on.

ABSTRACT OF THE DISSERTATION

Breaking Computational Barriers to Perform Time Series Pattern Mining at Scale and at the Edge

by

Zachary Pierce Zimmerman

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2019
Dr. Philip Brisk, Co-Chairperson
Dr. Eamonn Keogh, Co-Chairperson

Uncovering repeated behavior in time series is an important problem in many domains such as medicine, geophysics, meteorology, and many more. With the continuing surge of smart/embedded devices generating time series data, there is an ever growing need to perform analysis on datasets of increasing size. Additionally, there is an increasing need for analysis at low power edge devices due to latency problems inherent to the speed of light and the sheer amount of data being recorded. The matrix profile has proven to be a tool highly suitable for pattern mining in time series; however, a naive approach to computing the matrix profile makes it impossible to use effectively in both the cloud and at the edge. This dissertation shows how, through the use of GPUs and machine learning, the matrix profile is computed more feasibly, both at cloud-scale and at sensor-scale. In addition, it illustrates why both of these types of computation are important and what new insights they can provide to practitioners working with time series data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Time series data is produced all around us. The devices producing this data are many and their purposes highly varied. For example, EEG data which records a person's brain activity can be used to diagnose and treat problems relating to the brain. A smart system that can monitor this activity could detect abnormalities like a stroke or seizure and notify medical personnel. This type of immediate detection and alerting can save lives. Earthquake warning systems are another example of this; smart systems are able to monitor seismic waves for earthquakes and alert people before the shaking begins. If Japan did not have a widespread early warning system in place for the massive earthquake in 2011 many more people could have died [35].

There will be an estimated 20.4 billion connected Internet of Things (IoT) devices in the world by 2020 [62]. Following current trends, the number of deployed sensors is expected to reach one trillion units by 2030 [24]. These devices are generating a tremendous amount of data; it is expected that IoT devices are already generating hundreds of zettabytes

of data over a year. This is too much data to process, and much of it remains unused and either deleted or archived in cold-storage [58].

Much of the data being collected is time series data. It monitors the health of bridges [22], it monitors weather conditions [41], it monitors behavior of wildlife [1], it monitors water quality in lakes and oceans [38], the list goes on and on.

Rather than discard or archive this data, never to be seen again, what if we can use it? Can we distill information from it before locking it away or discarding it? These kinds of problems are what motivated this dissertation, which is directed at increasing the size and scope of the time series pattern mining problems that we can solve.

As one example, earthquake cataloging systems currently utilize a significant amount of human labeling and analysis. Imagine if we could perform this analysis and cataloging automatically at the sensor, without any human intervention. Providing a dynamic, up-to-date catalog many times larger than scientists currently have available, while detecting events which are not spotted by humans. This would provide resources for additional discoveries in seismology and would allow scientists access to a more complete picture of seismic activity on our planet.

There are similar use cases in other domains. Imagine a hospital with ECG analysis available on medical equipment which enables automatic detection of anomalies in heart rhythm and notification of appropriate medical personnel. Automated discovery of these anomalies would provide doctors with a more complete picture and allow more confidence in their diagnoses.

By using the methods developed in this work, the kinds of applications mentioned above are no longer intractable from a computational perspective.

The rest of this work is organized as follows: first, Chapter 2 introduces relevant background information. Then, Chapter 3 shows how high performance computing and accelerators can be leveraged to perform time series pattern mining quickly. Chapter 4 shows how we can push the limits of pattern mining even further using cloud computing. Chapter 5 shows how we can leverage the methods introduced in the previous chapters to produce an even faster, approximate pattern mining solution which can support streaming data and can be run on low power devices. Chapter 6 provides a preview of directions that this research could take in the future. Finally, Chapter 7 gives some concluding remarks.

Chapter 2

Background

2.1 The Matrix Profile

In recent years, the Matrix Profile (MP) [109] has proliferated as a useful tool which can provide insights into time series behavior. In particular the MP is effective in finding shape-based similarities in time series. Once the MP is computed, it enables motif discovery [109], discord discovery [109], chain discovery [112], segmentation [37], and many other useful time-series analyses with very little overhead. The utility of the MP has been discussed at length in prior works; this work focuses on the various ways of computing the MP in a highly scalable manner, enabling the computation of the MP on streaming data and low-power hardware, as well extending and modifying the definition of the MP to contain various other useful pieces of information.

The MP is easy to describe algorithmically, see [109] and Chapter 3 for a formal algorithmic definition. However, implementations of the MP can vary tremendously in terms of scalability, and the scalable versions of MP algorithms are significantly more complex.

In this work, an architecture aware implementation for graphics processing units (GPUs) to compute the MP is described in detail, along with its applications to seismology and other domains that depend on the scalability this implementation provides. This GPU implementation can be used in various ways. For example, it can be modified and used as a subroutine in a distributed cloud-scale deployment of the MP capable of performing 10^{18} comparisons in under 24 hours on 40 GPUs. This fast implementation can also be used to produce an even faster, approximate MP using machine learning. Chapter 3 begins with an overview of the MP and its utility in Section 3.1.

2.2 GPUs

The Graphics Processing Unit, or GPU, is "especially well-suited to address problems that can be expressed as data-parallel computations" [27]. It has its own memory, and it can launch multiple threads in parallel. Many parallel architectures use a Single Instruction Multiple Data (SIMD) architecture. For GPUs, the term Single Instruction Multiple Thread (SIMT) is also used. On a GPU, groups of threads process the same set of instructions on multiple data values in parallel, this is the SIMD/SIMT paradigm. Many GPU architectures exist, most notably Nvidia GPUs, which utilize the proprietary CUDA architecture; AMD GPUs which utilize the OpenCL toolchain, and GPUs which coexist on a CPU die (e.g. Intel HD Graphics). While there is certainly potential in working with all of these architectures, this work will focus on the utilization of Nvidia GPUs. The CUDA architecture used by Nvidia GPUs includes a system architecture, runtime, compiler, and programming language. Unless otherwise noted, the GPU terminology used in this work refers to components of the

CUDA architecture, while similar concepts exist in the other architectures, the terminology used can be different.

The threads on a GPU are managed in thread blocks which execute on the GPU's Streaming Multiprocessors (SMs) in groups of 32 threads. These threads are backed by a group 32 CUDA cores known as a warp. GPUs utilize a two-level scheduling policy where thread blocks are scheduled onto SMs, and each SM handles the scheduling of threads in that block using its own scheduler, known as a warp scheduler. SMs can have a variable number of cuda cores and GPUs can have a variable number of SMs depending on the model and architecture of the GPU. Threads in a thread block can cooperate with each other through shared local resources (cache and shared memory on the SM). Threads running in a warp execute mostly in lockstep, to maximize temporal and spacial locality with respect to the instruction and data pipelines.

A CUDA function is known as a kernel. A kernel consists of host (CPU) code (for launching the kernel and moving data to the GPU) along with device code which executes only on the GPU. When we launch a kernel, we can specify the number of blocks and the number of threads in each block to run on GPU. Currently, Nvidia allows launching at most 1024 threads within a block. Many blocks can be launched at once, but if there are too many blocks not all can be scheduled at once. There is a tradeoff between launching many threads that do less work versus launching few threads which do more work and different applications fall on different sides of the spectrum.

Nvidia GPUs are manufactured in various product lines to serve different markets where they are demanded:

- **GeForce: Consumer and Enthusiast Graphics Processing**: e.g. Real-time rendering for video games (Deep Learning)
- **Quadro: Commercial Graphics Processing**: e.g. Game Development, Rendering, Video Editing/Encoding
- **Tesla: Computation and Scientific Computing**: e.g. simulation, dynamics, high precision compute

The Tesla and Quadro lines tend to be more expensive than the consumer graphics (GeForce). One of the main advantages of Tesla GPUs is that they have significantly more double precision floating point units which enable them to perform high precision computation (required by many scientific computing applications) much faster than other Nvidia GPUs.

For further resources on the CUDA architecture and parallel programming on GPUs, please see [52].

2.3 Cloud Technology

In recent years, cloud technology has emerged as a significant resource for solving computational problems at scale. There are many reasons for this including:

- **Availability of Modern Hardware**: Cloud providers will stock more recent hardware. The hardware is upgraded by the provider, which means that the user will almost always have access to recent hardware, instead of needing to host a machine themselves and upgrade it every few years.

- **Elasticity and Scalability**: Users can pay for as much or as little as they need, they can automatically scale their application as demand increases or decreases.
- **Cloud Ecosystems and Technology**: As cloud platforms become more ubiquitous, they are offering more tools that make it easier for users to deploy applications in the cloud. These tools range in application from Data Ingestion, Data Storage, and even cutting edge ML and AI services.

Additionally, many cloud providers have adopted technologies used by the scientific community and made them available in the cloud. Currently, Amazon Web Services (AWS) offers access to instances with attached GPUs or FPGAs. Up to 8 GPUs can be attached to a single instance. Google Cloud Platform (GCP), offers access to the proprietary Google TPU for machine learning workloads, while also offering access to GPUs.

Many cloud platforms offer access to preemptible versions of their instances. A preemptible instance can be shut down by a cloud provider with very little notice during times of high demand. Applications running on preemptible instances must be fault tolerant otherwise an entire computational job could fail when the instance is preempted. These preemptible instances are offered at much cheaper rates compared to dedicated, non-preemptible instances. Essentially, users of preemptible instances are scraping the 'bottom of the barrel' in terms of compute resources. These preemptible instances are the resources nobody was willing to pay full price for and so the cloud provider is willing to sell them to the highest bidder. Applications that utilize preemptible instances can be many times cheaper than those that do not.

2.4 Matrix Profile Limitations

While the MP will work in *many* cases, it is not a panacea; there are cases where it will fail to provide useful insights. The MP is useful for finding *shape* based information in time series data. Sometimes time series data will not have useful shape based information causing the MP and related techniques to fail. This is not always possible to determine beforehand, and care should be taken in deploying the matrix profile at scale on a dataset which is untested for shape based information.

Chapter 3

The Matrix Profile & GPU-STOMP

3.1 Background

Time series motifs are approximately repeated subsequences found within a longer time series. While time series motifs have been in the literature for fifteen years [25], they recently have begun to receive significant attention *beyond* the data mining community. In recent years, they have been applied to a wide variety of problems, which include understanding the network of genes affecting the locomotion of the *C. elegans* nematode [56], severe weather prediction [60], and cataloging speech pathologies in humans [11].

Although significant progress has been made in how we score, rank, and visualize motifs, discovering them in large datasets remains a computational bottleneck. In this chapter, the reader will learn how we can significantly improve the scalability of exact motif

discovery both by leveraging GPU hardware and also by modifying the recently introduced STAMP algorithm [109]. The STAMP algorithm computes time series subsequence *joins* with an *anytime* algorithm [109]. Our key observations follow below:

- The solution to the full exact 1NN time series join can be converted to the exact solution for any definition of time series motif [65] with only trivial extra effort. Moreover, full exact 1NN time series join also yields the exact solution for time series discords, a popular definition for anomalies in time series [23].

- The anytime property of STAMP may be useful to some users. However, an anytime solution discards critical temporal and special locality which can be exploited to produce a faster algorithm on modern hardware and coprocessors. Additionaly, as we will explain below, in seismology, which is one of the the domains motivating this work, it is not required or helpful. As we will demonstrate, if we forego this property, we can compute motifs many orders of magnitude faster than STAMP.

By maintaining the "STAMP" theme introduced in [109], we call our faster algorithm **STOMP**, Scalable Time series Ordered-search Matrix Profile and its GPU-accelerated version **GPU-STOMP**.

In this chapter, we show that GPU-STOMP allows us to significantly expand the limits of scalability. We demonstrate the scalability of our ideas by extracting motifs from a dataset with one hundred and forty-three million subsequences. This requires computing (or admissibly pruning) 10,224,499,928,500,000 pairwise Euclidean distance values (i.e. more than ten quadrillion comparisons). If each Euclidean distance calculation took one microsecond, a brute force algorithm would require 324 years . An optimized implemen-

tation of GPU-STOMP (see Chapter 4) can compute this in only a few hours on a recent GPU. We recognize that a few GPU hours seems like significant computational time, but it is important to note that this data represents 83 days of continuous seismology recording at 20Hz. Therefore, even at this massive scale, our algorithm is much faster than real time. Figure 3.1 previews a pair of repeating earthquake sequences, which is essentially a time series motif [25], discovered by our algorithm in a seismologic dataset.

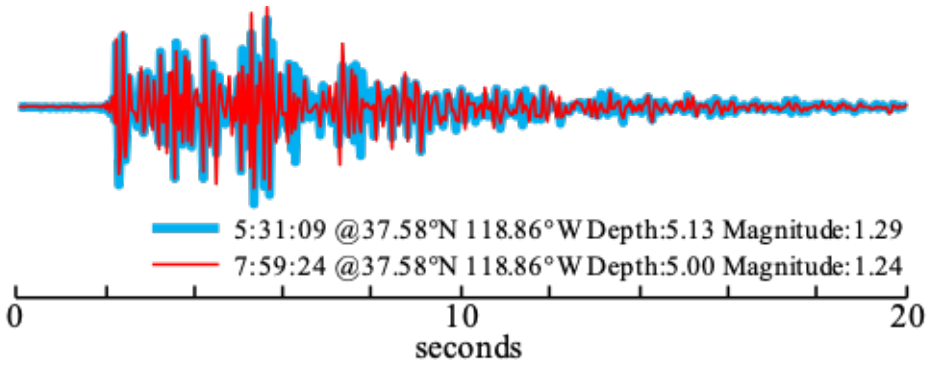

Figure 3.1: A pair of repeating earthquake sequences (motifs) we discovered from seismic data recorded at a station near Mammoth Lakes on February 17th, 2016. One occurrence (fine/red) is overlaid on top of another occurrence (bold/blue) that happened hours earlier. (best viewed in color).

Here, the two occurrences are very similar despite happening 148 minutes apart. Although the geophysics of earthquakes indicates that in principle we could see similar events millennia apart, we are unfortunately limited to the few decades humans have been recording such data (see Figure 3.13).

3.1.1 Motif Discovery Background and Matrix Profile

Motif discovery for time series was introduced in 2003 [25] (although the classic paper of Agrawal, Faloutsos and Swami foreshadows motifs by computing all-pair similarity for time series [2]). Since then, it has increased in research activity. One critical direction has been applying motifs to solve problems in a wide variety of domains such as bioinformatics [56], speech processing [11], robotics, human activity understanding [98][102], severe weather prediction [60], neurology, and entomology [65]. The other key research focus has been in the extensions and generalizations of the original work, especially in the attempts to improve scalability [55][65]. These attempts to improve the scalability of motif discovery fall into two broad classes; approximate and exact motif discovery [55][63][65].

Clearly approximate motifs can be much faster to compute (See Chapter 5), and this may be useful in many domains. However, there are certain problems spaces in which the risk of false negatives is unacceptable. Consider seismology, a domain in which false negatives could affect public policy, change insurance rates for customers, and conceivably cost lives by allowing a dangerous site to be developed for dwellings.

Beyond being exact, the proposed approach has many advantages that are not shared by rival methods:

- The proposed method is simple and parameter-free: In contrast, other methods typically require building and tuning spatial access methods and/or hash functions [25][55][57][63][98][102][110].

- It is space efficient: Our algorithm requires an inconsequential space overhead, just linear in the time series length, with a small constant factor. In particular, we avoid the need to actually explicitly extract the individual subsequences [25][63][65], something that would increase the space complexity by two or three orders of magnitude.

- It is incrementally maintainable: Having computed motifs for a dataset, we can incrementally update the best motifs very efficiently if new data arrives [109].

- It can leverage hardware: As we show below, our algorithm is embarrassingly parallelizable on multicore processors.

- Our algorithm has time complexity that is constant in subsequence length: This is a very unusual and desirable property; virtually all time series algorithms scale poorly as the subsequence length grows (the classic curse of dimensionality) [25][55][57][63] [98][110].

- Our algorithm takes deterministic time, dependent on the data's length, but completely independent of the data's structure/ noise level etc. This is also unusual and desirable property for an algorithm in this domain. For example, even for a fixed time series length, and a fixed subsequence length, all other algorithms we are aware of can radically different times to finish on two (even slightly) different datasets [25][55][57][63][98][110]. In contrast, given only the length of the time series, we can predict precisely how long it will take our finish in advance.

Virtually every time series data mining technique has been applied to the motif discovery problem, including indexing [55][104], data discretization [25], triangular-inequality

pruning [65], hashing [98][102][110], early abandoning, etc. However, all these techniques rely on the assumption that the intrinsic dimensionality of time series is much lower than the recorded dimensionality [25][98][102][105][110]. This is generally true for data such as short snippets of heartbeats and gestures, etc.; however, it is not true for seismographic data, which is intrinsically high dimensional. To ascertain this, we performed a simple experiment.

We measured the Tightness of Lower Bounds (TLB) for three types of data, using the two most commonly used dimensionality reduction representations for time series, DFT and PAA. Additionally, PAA is essentially equivalent to the Haar wavelets for this purpose [105]. The TLB is defined as:

$$TLB = \frac{LowerBoundDist(A, B)}{TrueEuclideanDist(A, B)} \tag{3.1}$$

It is well understood that the TLB is near perfectly (inversely) correlated with wall-clock time, CPU operations, number of disk access or any other performance metric for similarity search, all-pair-joins, motifs discovery, etc. [105]. As the mean TLB decreases, we quickly degrade to simple brute-force search. The absolute minimum value of TLB is dependent on the data, the search algorithm, and the problem setting (main-memory based vs disk based). However as [105] demonstrates, lower bound values less than 0.5 generally do not "break even."

Figure 3.2 shows unambiguous results. There is some hope that we could avail current speed-up techniques when considering (relatively smooth and simple) short snippets of ECGs, but there is little hope that the noisy and more complex human activity would

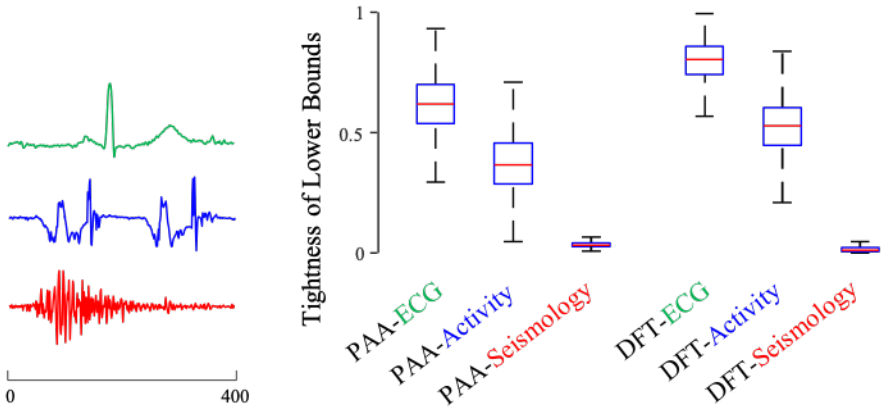

Figure 3.2: left) Samples from three datasets, ECG, Human Activity, and Seismology (available in [100]). right) The tightness of lower bounds, averaged over 10,000 random pairs, using PAA and DFT.

yield to such optimizations, and there is no hope that anything currently in the literature will help with seismological data. This claim is further proven in our detailed experiments in Section 3.3.

Even if we ignore this apparent death-knell for indexing/spatial access techniques, we could still dismiss them for other reasons, including memory considerations. As demonstrated in 3.2, a critical property of STOMP is that it does not need to explicitly extract the subsequences, which is unlike the indexing/spatial access methods. For example, consider a time series of length 100 million, with eight bytes per value, requiring 0.8 GB. Our algorithm requires an overhead of seven other vectors of the same size (including the output), for an easily manageable total of 6.4 GB (if memory was a bottleneck, we could reduce this by using reduced precision vectors or compression). However, any indexing algorithm that needs to extract the subsequences will increase memory requirements by at least $O(d)$, where d is the reduced dimensionality used in the index [57]. Given that d may be 20

or greater, this indicates the memory requirements grow to at least 16 GB. With such a large memory footprint, we are almost certainly condemned to a random-access disk-based algorithm, dashing any hope of any speedup.

A related advantage of our framework is that we can choose the subsequence length just prior to performing the motif discovery. In contrast, any index-based technique must commit to a subsequence length before building the index, and it could take hours/days to build the data structure before any actual searching could begin [78][105]. If such an index is built to support subsequences of say length 200, it cannot be used to join subsequences of length 190 or 205, etc. (See Section 1.2.3 of [78]). Thus, if we change our mind about the length of patterns we are interested in, we are condemned to a costly rebuilding of the entire index. It is difficult to overstate the utility of this feature. In section 3.3.8, we will demonstrate how we can use STOMP to explore the behavior of a penguin. At the beginning of this case study, we had no idea of what time frame the penguin's behavior might be manifest. However, with no costly index to build, we simply tried a few possible lengths until it was obvious that we found a reasonable value.

3.1.2 Seismological Background

While the Matrix Profile algorithms are completely general and can be applied to any domain, seismological data is of particular interest to us, due to its sheer scale and importance in human affairs.

In the early 1980s, it was discovered that in the telemetry of seismic data recorded by the same instrument from sources in given region, there will be many similar seismograms [36]. Geller and Mueller [36] have suggested that, "The physical basis of this clustering

is that the earthquakes represent repeated stress release at the same asperity, or stress concentration, along the fault surface." These patterns are called "repeating earthquake sequences" in seismology and correspond to the more general term "time series motifs." Figure 3.1 shows an example of a repeating earthquake sequence pair from seismic data.

A more recent paper notes that many fundamental problems in seismology can be solved by joining seismometer telemetry in locating these repeating earthquake sequences [110], which includes the discovery of foreshocks, aftershocks, triggered earthquakes, swarms, volcanic activity, and induced seismicity. However, the paper further notes that an exact join with a query length of 200 on a data stream of length 604,781 requires 9.5 days. Their solution, a transformation of the data to allow LSH based techniques, does achieve significant speedup, but at the cost of false negatives and necessary careful parameter tuning. For example, [109] notes that they need to set the threshold to precisely 0.818 to achieve decent results. While we defer a full discussion of experimental results to 3.3, the ideas introduced in this chapter can reduce the quoted 9.5 days for exact motif discovery from a dataset of size 604,781 to less than one minute, without tuning any parameters and also guaranteeing that false negatives will not occur.

It is vital to note that this kind of speed up really is game changing in this domain. It allows seismologists to quickly identify or detect earthquakes that are identical or similar in location without needing trilateration, and it can also provide useful information on relative timing and location of such events [5][49][50].

More controversially, some researchers have suggested that the slow slip on the fault accompanying non-volcanic tremors (a sequence of Low Frequency Earthquakes, many

of which are repeated) may temporarily increase the probability of triggering a large earthquake. Therefore, detecting and locating these repeating LFEs allows more accurate short-term earthquake forecasting [49].

Finally, we note that seismologists have been early adopters of GPU technology [61] and other high performance computing paradigms. However, their use of this technology has been limited to similarity search, not motif search.

3.1.3 Developing Intutions for the Matrix Profile

Unlike other motif/anomaly discovery systems, the matrix profile computes a score for every subsequence in the dataset. Here, we take the time to give some examples to demonstrate the utility of this more comprehensive annotation of data. We begin by considering the New York Taxi dataset of [79].

As shown in Figure 3.3.top, the data is the normalized number of NYC taxi passengers over 10 weeks, October 1st to December 15th 2014. The authors show this dataset to demonstrate the versatility of their "Attention Prioritization" technique for finding unusual patterns [79][9]. In essence, they transform the data (not shown here) in a way to make the discovery of anomalies easier. They note that Thanksgiving, on Thursday, November 27th, can be considered an "anomaly" in this dataset, since the patterns of travel apparently change during this important US holiday.

We computed the matrix profile for this dataset, with a subsequence length of one and a half days. As Figure 3.3.middle shows, the matrix profile peaks at the location that indicates Thanksgiving. However, there are additional observations that we can make with the matrix profile. There is a secondary anomaly occurring on Sunday, November 2nd;

there appears to be a spike in taxi demand at about 2:00 am. With a little thought, we realize this is exactly the hour in which daylight saving time is observed in the US. Setting the clock back one hour gives the appearance of doubling the normal demand for taxis at that hour. There is arguably a third anomaly in the dataset, with a more subtle, but still significant peak at October 13th. This day corresponds to Columbus Day. This holiday is all but ignored in most of the US, but it is still observed in New York, which has a strong and patriotic Italian community.

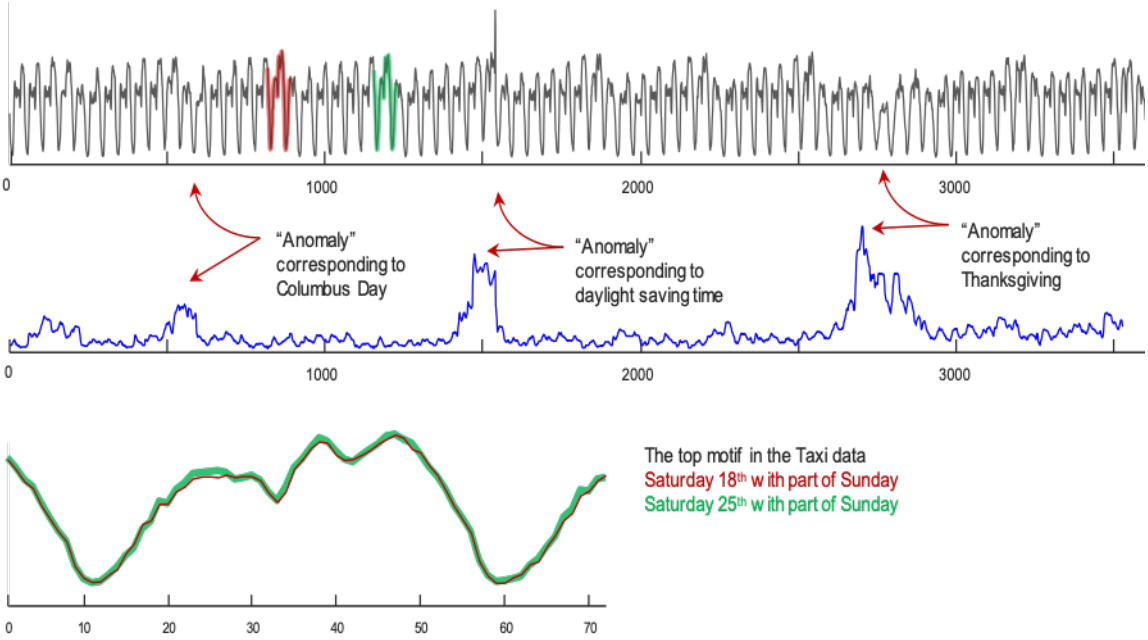

Figure 3.3: top) Normalized number of NYC taxi passengers over 10 weeks [79][9]. middle) The matrix profile produces high values where the corresponding subsequences are unusual. bottom) The top motif corresponds to two consecutive Saturdays.

In Figure 3.3.bottom, we show the top-1 motif from the dataset, which is extremely well conserved. In many natural datasets, for example the circadian rhythm of an animal, the best motifs are typically exactly twenty-four hours apart (a phenomenon known as

persistence). However, because this motif's two occurrences are exactly seven days apart, the importance of artificial divisions of the calendar on human behaviors becomes apparent. It is possible that the regions of lower conservation with the motif are also telling. For example, from 24 to 26 (about 10 to 11am), the motif corresponding to the 25th (green/bold) is a little higher than the previous week. It was lightly raining (about 0.12 inches) at the time, which may explain the slightly higher taxi demand in the late morning.

3.1.4 Notatition and Definitions

While we mostly follow the framework introduced in [109], for completeness we review all necessary definitions. We begin by defining the data type of interest, time series:

Definition 1 *A time series T is a sequence of real-valued numbers ti: $T = t_1, t_2, ..., t_n$, where n is the length of T.*

We are interested in local, not global properties of time series. A local region of time series is called a subsequence.

Definition 2 *A subsequence $T_{i,m}$ of a time series T is a continuous subset of the values from T of length m, which begin at position i. Formally, $T_{i,m} = t_i, t_{i+1}, ..., t_{i+m-1}$, where $1 \leq i \leq n - m + 1$.*

We can take a subsequence and compute its distance to all subsequences in the same time series. This is called a distance profile.

Definition 3 *A distance profile Di of time series T is a vector of the Euclidean distances between a given query subsequence Ti,m and each subsequence in time series T. Formally,*

$D_i = [d_{i,1}, d_{i,2}, ..., d_{i,n-m+1}]$, *where* $d_{i,j}(1 \leq i, j \leq n-m+1)$ *is the distance between* $T_{i,m}$ *and* $T_{j,m}$.

We assume that the distance is measured by Euclidean distance between z-normalized subsequences [105].

We are interested in finding the nearest neighbors of all subsequences in T, as the closest pairs of this are the classic definition of time series motifs [25][65]. Note that by definition, the *ith* location of distance profile D_i is zero, and it is close to zero just before and after this location. Such matches are defined as trivial matches [65]. We avoid such matches by ignoring an "exclusion zone" of length $\frac{m}{4}$ before and after the location of the query. In practice, we simply set $d_{i,j}$ to infinity for $(i - \frac{m}{4} \leq j \leq i + \frac{m}{4})$ while evaluating D_i.

We use a vector called matrix profile to represent the distances between all subsequences and their nearest neighbors.

Definition 4 *A matrix profile* P *of time series* T *is a vector of the Euclidean distances between each subsequence* $T_{i,m}$ *and its nearest neighbor (i.e. the closest match) in time series* T. *Formally,* $P = [min(D_1), min(D_2), ..., min(D_{n-m+1})]$, *where* $D_i(1 \leq i \leq n-m+1)$ *is the distance profile* D_i *of time series* T.

We call this vector a matrix profile, since it could be computed by using the full distance matrix of all pairs of subsequences in time series T, and evaluating the minimum value of each column (although this method is naïve and space-inefficient). Figure 3.4

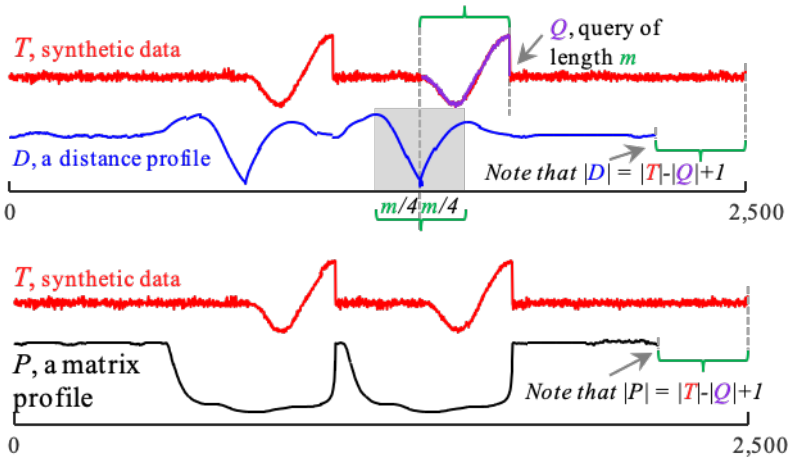

Figure 3.4: top) One distance profile (Definition 3) created from a random query subsequence Q of T. If we created distance profiles for all possible query subsequences of T, the element-wise minimum of this set would be the matrix profile (Definition 4) shown at (bottom). Note that the two lowest values in P are at the location of the 1st motif [25][65].

illustrates both a distance profile and a matrix profile created on the same raw time series T.

It is important to note that the full distance matrix is symmetric: D_i is both the *ith* row and the *ith* column of the full distance matrix. Figure 3.5 shows this more concretely.

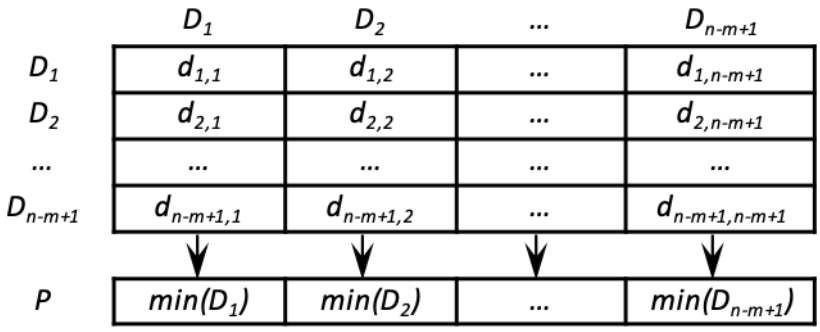

Figure 3.5: An illustration of the relationship between the distance profile, the matrix profile and the full distance matrix. For clarity, note that we do not actually create the full distance matrix, as this would have untenable memory requirements.

The *ith* element in the matrix profile P indicates the Euclidean distance from subsequence $T_{i,m}$ to its nearest neighbor in time series T. However, it does not indicate the location of that nearest neighbor. This information is recorded in a companion data structure called the matrix profile index.

Definition 5 *A matrix profile index I of time series T is a vector of integers: $I = [I_1, I_2, ...I_{n-m+1}]$, where $I_i = j$ if $d_{i,j} = min(D_i)$.*

By storing the neighboring information in this manner, we can efficiently retrieve the nearest neighbor of query $T_{i,m}$ by accessing the *ith* element in the matrix profile index.

As presented, the matrix profile is a self-join [109]: for every subsequence in a time series T, we find its (non-trivial-match) nearest neighbor within the same time series. However, we can trivially generalize the MP to be an AB-join [109]: for every subsequence in a time series A, record information about its nearest neighbor in time series B. Note that A and B can be of different lengths, and generally, AB-join $\neq$ BA-join.

To briefly summarize this section, we can create two Meta time series, the matrix profile and the matrix profile index, to annotate a time series T with the distance and location of all its subsequences' nearest neighbors within T. As the reader may already have realized, the smallest pair of values in the matrix profile correspond to the best motif pair by the classical definition [55][65][25], and the corresponding values in the matrix profile index indicate the location of the motif. Moreover, as both [109][65] argue, the top-k motifs, range motifs, and any other reasonable variant of motifs can trivially be computed from the information in the matrix profile, which is the focus of the remainder of this work.

3.2 Algorithms

In this section, we begin by demonstrating that we can improve upon the STAMP algorithm [109] to create the much faster STOMP algorithm. Then we demonstrate that the structure of STOMP lends itself to porting to GPUs.

3.2.1 The STOMP Algorithm

As explained below, STOMP is similar to STAMP [109] in that it can be viewed as highly optimized nested loop searches with repeating calculations of distance profiles in the inner loop. However, while STAMP must evaluate the distance profiles in a random order (to allow its anytime behavior), STOMP performs an *ordered* search. By exploiting the locality of these searches, we can reduce the time complexity by a factor of $O(\log n)$. Before we explain the details of the algorithm, we first introduce a formula to calculate the z-normalized Euclidean distance $d_{i,j}$ of two time series subsequences $T_{i,m}$ and $T_{j,m}$ by using their dot product, QTi, j:

$$d_{i,j} = \sqrt{2m(1 - \frac{QT_{i,j} - m\mu_i\mu_j}{m\sigma_i\sigma_j})} \tag{3.2}$$

Here m is the subsequence length, μ_i is the mean of $T_{i,m}$, μ_j is the mean of $T_{j,m}$, σ_i is the standard deviation of $T_{i,m}$, and σ_j is the standard deviation of $T_{j,m}$.

The technique introduced in [78] allows us to obtain the means and standard deviations with $O(1)$ time complexity; thus, the time required to compute $d_{i,j}$ depends only on the time required to compute $QT_{i,j}$. Here, we claim that $QT_{i,j}$ can also be computed in $O(1)$ time, once $QT_{i-1,j-1}$ is known.

Note that $QT_{i-1,j-1}$ can be decomposed as the following:

$$QT_{i-1,j-1} = \sum_{k=0}^{m-1} t_{i+k-1} t_{j+k-1} \tag{3.3}$$

and $QT_{i,j}$ can be decomposed as the following:

$$QT_{i,j} = \sum_{k=0}^{m-1} t_{i+k} t_{j+k} \tag{3.4}$$

Thus we have:

$$QT_{i,j} = QT_{i-1,j-1} - t_{i-1} t_{j-1} + t_{i+m-1} t_{j+m-1} \tag{3.5}$$

Therefore, our claim is proved.

Figure 3.6 visualizes the algorithm. Based on Equation 3.2, we can map the distance matrix computation in 3.5 (also shown in 3.6.left) to its corresponding dot product matrix (shown in 3.6.right).

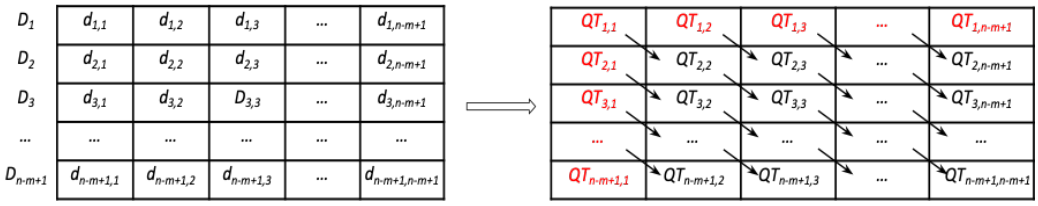

Figure 3.6: Mapping the computation of the distance matrix (left) to the computation of its corresponding dot product matrix (right).

The arrows in 3.6.right show the data dependency indicated by 3.5: once we have $QT_{i-1,j-1}$, we can compute $QT_{i,j}$ in $O(1)$ time. However, note that 3.5 does not

apply to the special case when $i = 1$ or $j = 1$ (the first row and the first column of 3.6.right, shown in red). This problem is easy to solve: we can pre-compute the dot product values in these two special cases with a FFT, as shown in Table 3.1. Concretely, we use $SlidingDotProduct(T_1, m, T)$ to calculate the first dot product vector: $QT_1 = [QT_{1,1}, QT_{1,2}, ..., QT_{1,n-m+1}] = [QT_{1,1}, QT2, 1, ..., QT_{n-m+1,1}]$. The dot product vector is stored in memory and used as needed.

Table 3.1: Calculate Sliding Dot Products with FFT.

Procedure *SlidingDotProducts*
Require: Q a query sequence, T a time series
1: $n := Length(T)$, $m := Length(Q)$
2: $Q_r := Reverse(Q)$
3: $Q_{ra} :=$ Append Q_r with $n - m$ zeros
4: $Q_{raf} := FFT(Q_{ra})$, $T_f = FFT(T)$
5: $QT = InverseFFT(ElementwiseMultiply(Q_{raf}, T_f))$
6: **return** $QT[m : n]$

After the first row and the first column in 3.6.right are computed, the rest of the dot product matrix is evaluated row after row. We are now in the position to introduce our STOMP algorithm in Table 3.2.

The algorithm begins in line 1 by computing the matrix profile length l. In line 2, the mean and standard deviation of every subsequence in T are pre-calculated. Line 3 calculates the first dot product vector QT with the algorithm in Table 3.1. In line 5, we initialize the matrix profile P and matrix profile index I. The loop in lines 6-13 calculates the distance profile of every subsequence of T in sequential order. Lines 7-9 update QT according to Equation 3.5. We update $QT[1]$ in line 10 with the pre-computed QT_{first} in

Table 3.2: The STOMP algorithm for computing the Matrix Profile

Procedure *STOMP*
Require: T a time series, m a subsequence length
1: $n := Length(T)$, $l := n - m + 1$
2: $\mu, \sigma := ComputeMeanStd(T, m)$ // see [78]
3: $QT := SlidingDotProduct(T[1:m], T), QT_{first} := QT$
4: $D := CalculateDistanceProfile(QT, \mu, \sigma, 1)$ {see Equation 3.2}
5: $P := D$, $I := ones$ {initialization of Matrix Profile P and Matrix Profile Index I}
6: **for** $i = 2$ **to** l **do**
7: **for** $j = l$ **downto** 2 **do**
8: $QT[j] := QT[j-1] - T[j-1] * T[i-q] + T[j+m-1] * T[i+m-1]$ {update dot product, see Equation 3.5}
9: **end for**
10: $QT[1] := QT_{first}[i]$
11: $D := CalculateDistanceProfile(QT, \mu, \sigma, i)$ {see Equation 3.2}
12: $P, I := ElementWiseMin(P, I, D, i)$
13: **end for**
14: **return** P, I

line 3. Line 11 calculates distance profile D according to 3.2. Finally, line 12 compares every element of P with D: if $D[j] < P[j]$, then $P[j] = D[j]$, $I[j] = i$.

The time complexity of STOMP is $O(n^2)$; thus, we have achieved a $O(\log n)$ factor speedup over STAMP [109]. Moreover, it is clear that $O(n^2)$ is optimal for any exact motif algorithm in the general case. The $O(\log n)$ speedup makes little difference for small datasets and for those with just a few tens of thousands of data points [25]. However, as we consider the datasets with millions of data points, this $O(\log n)$ factor begins to produce a very useful order-of-magnitude speedup.

To better understand the efficiency of STOMP, it is important to clarify that the time complexity of the classic brute force algorithm is $O(n^2 m)$. The value of m depends on the domain, but in Section 3.3.8, we consider real world applications where it is 2,000. Most techniques in the literature gain speedup by slightly reducing the n^2 factor; however, we

gain speedup by reducing the m factor to $O(1)$. Moreover, it is important to remember that the techniques in the literature can only reduce this n^2 factor if the data has a low intrinsic dimensionality (recall Figure 3.2), and the domain requires a short subsequence length. In contrast, the speedup for STOMP is completely independent of both the structure of the data and the subsequence length.

Despite this dramatic improvement, it still takes STOMP approximately 5-6 hours to process a time series of length one million. Can we further reduce the time?

It is important to note that the STOMP algorithm is extremely amenable to parallel computing frameworks. This is not a coincidence; the algorithm was conceived with regards to eventual hardware acceleration. Recall that the space requirement for the algorithm is only $O(n)$; there is no data dependency in the main inner loop of the algorithm (lines 7-9 of Table 3.2), so we can update all entries of QT in parallel. The evaluation of each entry in vectors D, P, and I in lines 11 and 12 are also independent of each other. In the next section, we will introduce a GPU-based version of STOMP, utilizing these observations to further speed up the evaluation of the matrix profile and thus motif discovery.

3.2.2 Porting STOMP to a GPU Framework

Note: Please refer to Section 2.2 for an explanation of GPU architecture and terminology. This section assumes the reader is familiar with these concepts.

A naïve GPU implementation of the STOMP algorithm in TABLE II can be decomposed into four steps:

- CPU copies the time series to GPU global memory.

- CPU launches GPU kernels to evaluate μ, σ, the initial QT, D, P and I.
- CPU iteratively launches GPU kernels to update QT, D, P, and I.
- CPU copies the final output (P and I) from GPU.

In the first step, the CPU copies time series T (input vector of Table 3.2) to the global memory of GPU. The time used to copy a time series of length 100 million takes less than a second, however it is important that not too much memory transfer between the host and the GPU takes place.

Note that in order to run the STOMP algorithm, we need to allocate space to store eight vectors in the GPU global memory: T, μ, σ, QT, QT_{first}, D, P and I. A double-precision time series of length 100 million is approximately 0.8GB, so the algorithm utilizes approximately 6.4GB global memory space. This is feasible for NVIDIA Tesla K40 and K80 cards; however, if the device used has less memory space available, we can split the time series into small sections and evaluate one section at a time with the GPU. See Chapter 4 for details on how this can be done.

In the second step, the CPU can launches GPU kernels to evaluate the vectors in parallel. The mean and standard deviation vectors in line 2 of Table 3.2 can be efficiently evaluated by CUDA Thrust [27] using parallel prefix-sum algorithms. The QT_{first} vector in line 3 can be generated in parallel by using cuFFT, the NVIDIA CUDA Fast Fourier Transform Library [28] in the SlidingDotProduct procedure from Table 3.1. We assign a total of $n - m + 1$ threads to evaluate QT_{first}, D, P, and I in lines 3-5 in parallel. The jth thread processes the jth entry of these vectors one by one.

Now that we have initialized QT, D, P, and I, we update them iteratively. In the third step, the CPU runs the outer loop in lines 6-13 of Table 3.2 iteratively. In every iteration, the CPU launches a GPU kernel with $n-m+1$ threads, parallelizing the evaluation of QT, D, P, and I. As shown in Figure 3.7, the first thread reads $QT[1]$ from the pre-computed QT_{first} vector, while the second to the last threads evaluate their corresponding entry of QT using Equation 3.5.

Note that in contrast to the CPU STOMP algorithm, which uses only one vector QT to store both QT_{i-1} and QT_i, here we use two vectors to separate them. This is necessary because as the threads evaluate entries in QT in parallel, we need to avoid writing entries before they are read. A simple and efficient way to accomplish this is to create two vectors, QT_{odd} and QT_{even}. When the outer loop variable i in line 6 is even, the threads read from QT_{odd} and write to QT_{even}; when i is odd, the threads read data from QT_{even} and write to QT_{odd}. Following this, the threads evaluate D with Equation 3.2, and the jth thread updates P and I if $D[j] < P[j]$.

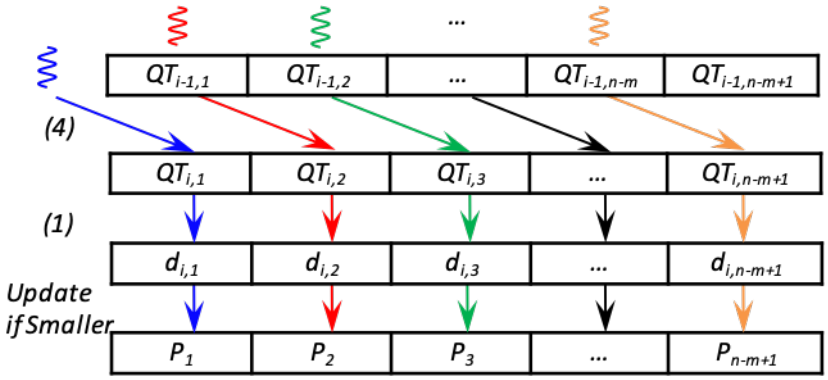

Figure 3.7: Division of work among threads in the third step of GPU-STOMP.

When all of the iterations are complete, we have reached the last step of GPU STOMP, where the CPU copies P and I back to the system memory.

3.2.3 Further Parallelizing STOMP with multiple GPUs

The parallelization scheme above is suitable if we only have one GPU device. Can we further reduce the processing time if there are two or more GPUs available?

Thus far, we have been using CPU to iteratively control the outer loop of the STOMP algorithm in Table 3.2. We start by computing the first distance profile (the first row) in Figure 3.5 and its corresponding QT vector. Then in each iteration, we compute a new row of the distance matrix in Figure 3.5, and maintain the minimum-so-far values of each column in vector P. When the iteration is complete, P becomes the exact matrix profile.

This outer loop computation can be further parallelized. Assume we have k independent GPU devices, and we also have $(n-m+1)/k = q$. We can then divide the distance matrix in Figure 3.5 into k sections: device 1 evaluates the $1st$ to the qth rows, device 2 evaluates the $(q+1)th$ to the $(2q)th$ rows, etc. Essentially, device k uses the parallelized version of SlidingDotProduct function in Table 3.1 to calculate $QT_{q(k-1)+1}$ and $D_{q(k-1)+1}$, then it evaluates the following $q-1$ rows iteratively. The k devices can run in parallel, and after the evaluation completes, we can simply find the minimum among all the k matrix profile outputs. In summary, we can achieve a k-times speed up by using k identical GPU devices.

By porting all the introduced techniques to Nvidia Tesla K80, which contains two GPU devices on the same unit, we are able to obtain the matrix profile and matrix profile

index of a seismology time series of length 100 million within 19 days. But there are even more things we can do to accelerate GPU-STOMP.

3.2.4 A Technique to Further Accelerate GPU-STOMP

Figure 3.7 showed the process to compute the ith row of the distance matrix in Figure 3.5 by $n - m + 1$ parallel threads. Recall that the distance matrix is symmetric; half of the distance computations can be saved if we only evaluate the ith to the last columns. We show this strategy in Figure 3.8.top.

However, note that it is desirable to maintain the $O(n)$ space complexity of our algorithm; if we move on to the $(i + 1)th$ row of Figure 3.5 without further processing, then $P_i = min(d_{1,i}, d_{2,i}, ..., d_{i,i})$, and it would no longer be updated. To correct this, it is necessary to launch another kernel after Figure 3.8.top is completed. The new kernel is shown in Figure 3.8.bottom.

Essentially, we have used an analogous reduction technique as in [44] to obtain $d_{min} = \min(d_{i,i+1}, d_{i,i+2}, ..., d_{i,m+n-1})$, which also is equivalent to $\min(d_{i+1,i}, d_{i+2,i}, ..., d_{n-m+1,i})$ as a result of symmetry. If $d_{min} < P_i$, we set $P_i = d_{min}$, so $P_i = \min(D_i)$. Although it is necessary to launch an additional kernel to process each row, which will require extra time, the extra time is still less than what is saved when handling large time series.

For example, this new technique reduced the time to process a time series of length 100 million from 19 days to approximately 12 days on NVIDIA Tesla K80. This indicates

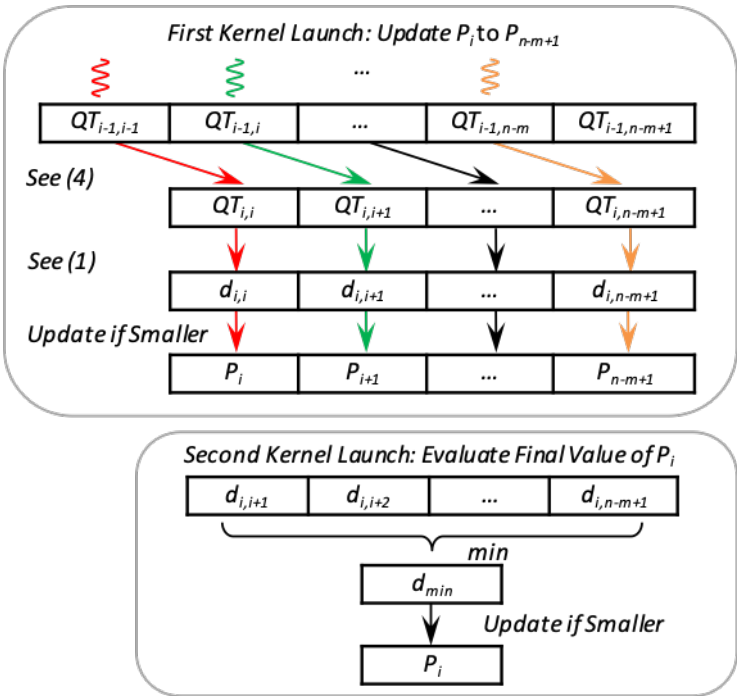

Figure 3.8: Modifying the third step of GPU-STOMP. top) Launch only $n - m - i + 2$ threads (instead of the $n - m + 1$ threads in Figure 3.7) this time at the ith iteration. bottom) Launch another kernel to evaluate the final value of P_i.

that it is possible to finish five quadrillion pairwise comparison of subsequences within 12 days.

Note that fewer and fewer threads are being launched in each iteration. To apply this new technique to multiple GPUs, it is necessary to ensure that each GPU is loaded with similar amount of work, so they will finish in similar time. Here, for NVIDIA Tesla K80, we computed the first $(n - m + 1)(1 - \frac{1}{\sqrt{2}})$ distance profiles with the first GPU and the last $\frac{n-m+1}{\sqrt{2}}$ distance profiles with the second GPU.

3.2.5 Breaking the Ten Quadrillion Pairwise Comparison Barrier

In the last section, we demonstrated a technique to use parallel threads to evaluate the rows of the distance matrix in Figure 3.5 iteratively. Note that to compute one row, the technique needs to launch two kernels, all threads need to be synchronized following the evaluation, and the corresponding QT vector needs to be updated in GPU global memory. As there are $n - m + 1$ rows in Figure 3.5, when n becomes large, the overhead for kernel launches, synchronization, and memory writes become nontrivial.

To make GPU-STOMP even faster, we need to modify the kernel to make it aware of the architectural considerations above. We denote this optimized, more architecture-aware version as **GPU-STOMPopt**. To help the reader better understand how the GPU-STOMPopt works, we will first show our initial optimization scheme in Figure 3.9, then further refine it in Figure 3.10 and Figure 3.11.

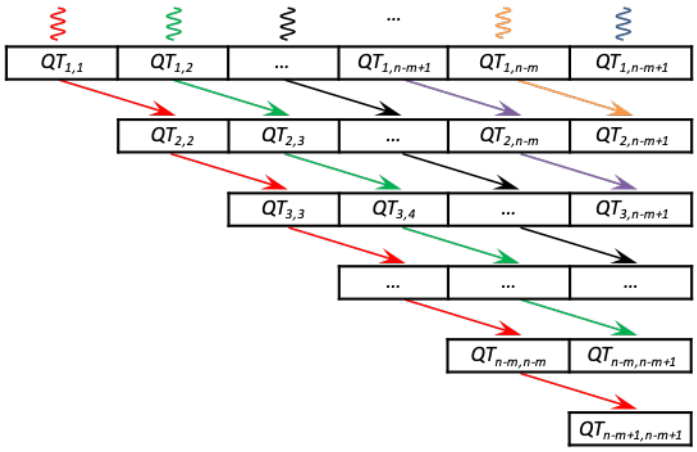

Figure 3.9: An optimization scheme for the the third step of GPU-STOMP. We only need to launch one kernel to evaluate all the rows of the distance matrix in Figure 3.5

Figure 3.9 shows our key scheme to save the kernel launch and thread synchronization time: instead of launching a kernel for every single row in Figure 3.5, we issue only one single kernel to generate the entire matrix profile. Note that based on the one-one correspondence between $d_{i,j}$ and $QT_{i,j}$ (as shown in Equation 3.2), we can convert the symmetric distance matrix computation into Figure 3.9, where we evaluate the upper-right half of the dot product matrix. Since the value of $QT_{i,j}$ is only dependent on $QT_{i-1,j-1}$ (according to Equation 3.5), the computation of each diagonal in Figure 3.9 is independent of any other diagonal. Thus, we assign $n - m + 1$ threads to compute these diagonals in parallel.

Once we obtain $QT_{i,j}$, we can easily evaluate $d_{i,j}$ based on Equation 3.2. Then we examine two elements of the matrix profile: if $d_{i,j} < P_i$, we set $P_i = d_{i,j}$; and if $d_{i,j} < P_j$, we set $P_j = d_{i,j}$. Note that as each thread in Figure 3.9 operates independently, multiple threads may attempt to update the same entry of the matrix profile at the same time. We need to use atomic operations to organize this. Essentially, we set a lock for each entry of the matrix profile. When multiple threads try to update the same matrix profile entry, they line up to get the lock, and perform an min operation in order. The reader may doubt that this can result in a significant cost of time, as it is possible that all threads can be lining up to update the same single matrix profile entry. However, in practice, we find that a large portion of these atomic operations are pruned from the calculation.

Assume we have twenty atomic operations lined up to update a matrix profile entry, which has an initial value of 6.81, with the following distance values in order:

0.6, 4.46, 1.99, 6.98, 2.29, 2.95, 7.05, 1.47, 6.04, 2.72, 2.31, 3.2, 6.25, 9.33, 0.27, 2.62, 2.00, 2.74, 6.67, 2.34.

Since the matrix profile entry keeps track of the minimum distance value, only two updates would be executed: 0.6 and 0.27. That is only 10% of this short sequence of data. Now let us randomly shuffle the data:

7.05, 2.29, 1.47, 0.27, 2.74, 2.95, 9.33, 2.34, 4.46, 2.00, 6.04, 2.72, 2.31, 3.2, 6.25, 6.98, 0.6, 2.62, 1.99, 6.67.

This time three updates would be executed: 2.29, 1.47, 0.27. That is only 15% of the data; so again, it is only a small portion.

Note that our toy example here is a very short data sequence. In practice, for most time series only less than 0.1% distance values end up smaller than their corresponding matrix profile elements. For example, for a random-walk time series of length one million, we executed on average only 39 atomic updates for each matrix profile entry; more than 99.996% of the atomic operations are pruned.

By implementing the optimization scheme shown in Figure 3.9, we have obtained about 3X speedup over GPU-STOMP for medium-size time series (i.e. with less than 4 million data points). However, as the time series gets even longer, less speedup is observed, as the time spent on atomic operations and global memory writes become nontrivial.

To solve this, we use two strategies to refine our optimization scheme in Figure 3.9:

The first strategy aims to accelerate each atomic write. As stated previously, multiple independent threads can be attempting to update the matrix profile at the same time, so we are using atomic operations to organize them. Note that when a matrix profile entry (which is a 64-bit double precision value) is updated, the corresponding matrix profile

index value (a 32-bit integer value) also needs to be updated. However, currently CUDA only supports atomic operations on either one single 32-bit value or one single 64-bit value. To tackle this, we initially set a lock on every entry of the matrix profile, and used a critical section to update both the matrix profile entry and the matrix profile index value when a thread gets the lock; however, this solution is not scalable with longer time series inputs. As a result, we turned to a better solution as shown in Figure 3.10. Instead of using a time-consuming critical section, we lower the precision of the matrix profile to 32 bits. We then combine the matrix profile and the matrix profile index into one double-precision vector in the global memory that can be atomically updated. For the ith entry of the double-precision vector, 32 bits are used to store the ith matrix profile value, and another 32 bits are used to store the ith matrix profile index.

This refinement strategy largely accelerated the speed for atomic operations. Note that the strategy will not result in large precision loss, as only the precision of the output is reduced; we are still using 64 bits to store all the intermediate results during the evaluation process.

The second strategy is to utilize the CUDA shared memory to ease the contention for global memory writes. The strategy, as shown in Figure 3.11, can be viewed as 2-level hierarchy of Figure 3.9. Here we define TPB as the number of threads per block in the CUDA kernel.

Different from Figure 3.9, in which each thread evaluates one single diagonal of the distance matrix, here we divide the distance matrix into k meta diagonals (as shown in Figure 3.11.a. A meta diagonal consists of TPB diagonals of the distance matrix [$(k -$

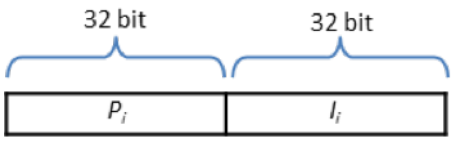

Figure 3.10: We reduced the matrix profile to 32 bits, then combined each matrix profile entry and its corresponding matrix profile index entry into a double-precision value to allow fast atomic updates.

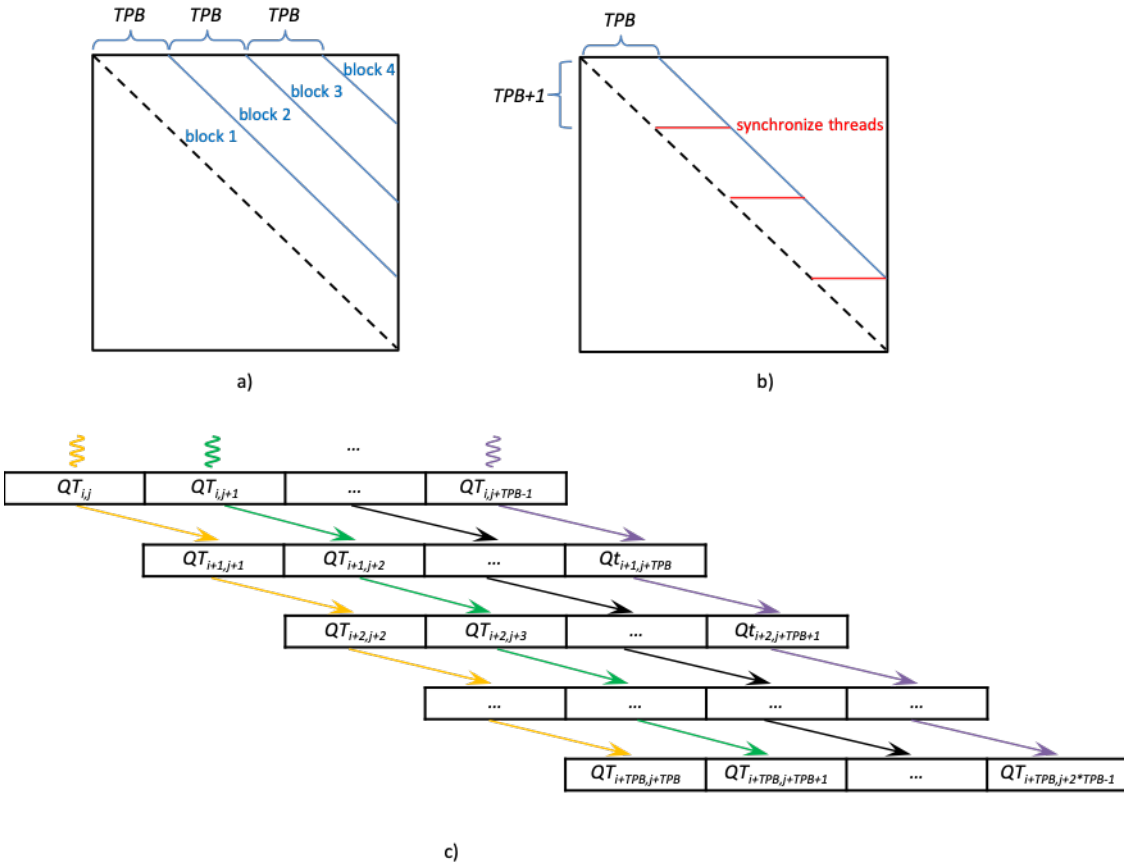

Figure 3.11: a) Each thread block evaluates one meta diagonal of the distance matrix. b) The parallelograms in a meta diagonal are evaluated iteratively by a thread block. c) The threads in a block evaluate diagonals of a parallogram in parallel.

$1) * TPB < n - m + 1 \leq k * TPB]$. Each meta diagonal is evaluated by one CUDA thread block. As shown in Figure 3.11.b, the thread block evaluates one parallelogram at

a time, managing a local copy of the matrix profile in its shared memory. The threads in a block (shown in Figure 3.11.c) work very similarly as those in Figure 3.9, except that they atomically update the shared memory instead of the global memory. After a parallelogram (Figure 3.11.b) is evaluated, all the threads in the block are synchronized. If any value in the shared memory is smaller than its corresponding entry in the global memory, the global memory is updated.

With this refinement strategy, the contention of atomic updates in Figure 3.9 is largely relieved. The original scheme in Figure 3.9 allowed a global memory location to be visited by all active threads in all the thread blocks (which can be as many as $n - m + 1$ threads) simultaneously. In contrast, with the refined scheme in Figure 3.11, the number of threads racing for a shared memory location cannot be larger than TPB, and a global memory location cannot receive more than k atomic update requests at the same time. This brings about a large performance gain.

Similar to GPU-STOMP, GPU-STOMPopt can easily be adapted to multiple GPUs as well. For example, to evenly divide the work for an Nvidia Tesla K80, we compute the odd (1st. 3rd, 5th, etc. from the left) meta diagonals in Figure 3.11.a with the first GPU, and the even (2nd, 4th, 6th, etc. from the left) meta diagonals in Figure 3.11.a with the second GPU. However, there are better ways of approaching splitting the work which will be covered in Chapter 4.

With all the optimization strategies, GPU-STOMPopt achieved more than 2X speedup over GPU-STOMP for large datasets. Concretely, it further reduces the time to process a time series of length 100 million from 12 days to about 4 days on Nvidia Tesla

K80. Furthermore, for the first time in the literature, we are able to process a time series of length 143 million, which is slightly more than ten quadrillion pairwise comparison of subsequences, within just 9 days.

It is important to note that with more modern GPUs this speedup is even more pronounced, because before the Maxwell architecture (the Tesla K80 is a Kepler architecture GPU, which is one generation prior to Maxwell) shared memory atomics were implemented in the instruction set, but not at the hardware level. More modern GPUs are many times faster than the K80 because their shared memory atomics are implemented in hardware [26]. Chapter 4 shows an evaluation of GPU-STOMPopt on newer hardware.

3.3 Empirical Evaluation

Although some parts of our experiments require access to a GPU, we have designed them so they can be reproduced easily. To allow for the reproduction of our experiments, we have constructed a webpage [100], which contains all datasets and code used in this chapter. We begin with a careful comparison to STAMP, which is obviously the closest competitor, and we consider more general rival methods later.

Unless otherwise noted, we used an Intel i7@4GHz PC with 4 cores to evaluate all the CPU-based algorithms; we used a server with two Intel Xeon E5-2620@2.4GHz cores and an Nvidia Tesla K80 GPU to evaluate GPU-STOMP and GPU-STOMPopt.

3.3.1 STAMP vs STOMP

We begin by demonstrating that STOMP is faster than STAMP, and also that this difference grows as we consider increasingly large datasets. Furthermore, we measure the gains made by using GPU-STOMP. In Table 3.3, we measure the performance of the three algorithms on increasingly long random walk time series with a fixed subsequence length 256.

Table 3.3: Time required for motif discovery with $m = 256$, varying n, for the three algorithms under consideration

	Runtimes (n)				
Algorithm	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}
STAMP	15.1 min	1.17 hours	5.4 hours	24.4 hours	4.2 days
STOMP	4.21 min	0.3 hours	1.26 hours	5.22 hours	0.87 days
GPU-STOMP	10 sec	18 sec	46 sec	2.5 min	9.25 min

Note that we choose m's length as a power-of-two only to offer the best case for (the FFT-based) STAMP; our algorithm is agnostic to such issues.

A recent paper on finding motifs in seismograph datasets also considers a dataset of about 2^{19} in length and reports taking 1.6 hours, which is approximately the same time it takes STOMP [110]. However, their method is probabilistic and allows false negatives (twelve of which were actually observed, after checking against the results of a 9.5 day brute-force search [110]). Moreover, it requires careful tuning of several parameters, and it does not lend itself to GPU implementation.

We wish to consider the scalability of even larger datasets with GPU-STOMP. However, in order to do so, we must estimate the time it takes the other two other algorithms. Fortunately, both of the other algorithms allow for an approximate prediction of

the time needed, given the data length n. To obtain the estimated time, we evaluated only the first 100 distance profiles of both STAMP and STOMP and multiplied the time used by $\frac{n-m+1}{100}$. In Table 3.4, we consider even larger datasets, one of which reflects the data used in a case study in Section 3.3.5.

Table 3.4: Time required for motif discovery with various m and various n, for the three algorithms under consideration

	Runtimes (m | n)	
Algorithm	2000 | 17,279,800	400 | 100,000,000
STAMP (estimated)	36.5 weeks	25.5 years
STOMP (estimated)	8.4 weeks	5.4 years
GPU-STOMP (actual)	9.27 hours	12.13 days

Note that the 100-million-length dataset is one hundred times larger than the largest motif search in the literature [55]. In all three algorithms under consideration, the time required is independent of the subsequence length m, which is desirable. This is demonstrated in Table 3.5, where we measure the time required with n fixed to 2^{17}, for increasing m.

Table 3.5: Time required for motif discovery with $n = 2^{17}$, varying m, for the three algorithms under consideration

	Runtimes (m)				
Algorithm	64	128	256	512	1,024
STAMP	15.1 min	15.1 min	15.1 min	15.0 min	14.5 min
STOMP	4.23 min	4.33 min	4.21 min	4.23 min	2.92 min
GPU-STOMP	10 sec	10 sec	10 sec	10 sec	10 sec

Note that the time required for the longer subsequences is slightly shorter. This is true since the number of pairs that must be considered for a time series join [109] is $(n-m+1)^2$, so as m becomes larger, the number of comparisons becomes slightly smaller.

3.3.2 GPU-STOMPopt Breaks the Ten Quadrillion Pairwise Comparison Barrier

In Table 3.6, we measure the performance of STAMP, STOMP, GPU-STOMP, and GPU-STOMPopt on increasingly long random walk time series with a fixed subsequence length 256. The shaded cells are duplicated from Table 3.3, but they are included for comparison. Note that while some numbers are estimated, as explained in the next section, we can predict the time and memory requirement of STAMP and STOMP very precisely (with less than 5% error) for large datasets.

Table 3.6: Time required for motif discovery with $m = 256$, varying n, for STAMP, STOMP, GPU-STOMP, and GPU-STOMPopt

Small Datasets

	Runtimes (n)				
Algorithm	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}
STAMP	15.1 min	1.17 hours	5.4 hours	24.4 hours	4.2 days
STOMP	4.21 min	0.3 hours	1.26 hours	5.22 hours	0.87 days
GPU-STOMP	10 sec	18 sec	46 sec	2.5 min	9.25 min
GPU-STOMPopt	8 sec	9 sec	17 sec	49 sec	2.93 min

Large Datasets

	Runtimes (n)		
Algorithm	17,279,800	100,000,000	143,000,000
STAMP	*36.5 weeks (est.)*	*25.5 years (est.)*	*51.2 years (est.)*
STOMP	*8.4 weeks (est.)*	*5.4 years (est.)*	*10.9 years (est.)*
GPU-STOMP	9.27 hours	12.13 days	*24.5 days (est.)*
GPU-STOMPopt	3.29 hours	4.51 days	9.33 days

3.3.3 STOMP vs State-of-the-Art Motif Discovery Algorithms

Beyond the independence of the subsequence length demonstrated in Table 3.5, all three matrix profile-based algorithms are also independent of the intrinsic dimensionality of the data, which is also desirable. To demonstrate this, we will compare to the recently introduced Quick-Motif framework [55] and the more widely known MK algorithm [65]. The Quick-Motif method was the first technique to perform an exact motif search on one million subsequences.

To level the playing field, we do not avail of GPU acceleration, but instead, we use the identical hardware (a PC with Intel i7-2600@3.40GHz) and programming languages for all algorithms. Note that for a fair comparison with STAMP [109], which is written in MATLAB, in Section 3.3.1, we measured the performance of STOMP based on its MATLAB implementation. However, because the two rival methods in this section (Quick-Motif and MK) are written in C/C++, here we measure the runtime of (the CPU version of) STOMP based on its C++ implementation.

We use the original author's executables [99] to evaluate the runtime of both MK and Quick-Motif. The reader may wonder why the experiments here are less ambitious than in the previous sections. The reason is that beyond time considerations, the rival methods have severe memory requirements. For example, for a seismology data with $m = 200$, $n = 2^{18}$, we measured the Quick-Motif memory footprint as large as 1.42 GB. In contrast, STOMP requires only 14MB memory for the same data, which is less than a hundredth of the footprint. If this ratio linearly interpolates, Quick-Motif would need more than half a terabyte of main memory to tackle the one hundred million benchmark, which is

infeasible. Moreover, for Quick-Motif, it is possible that a different dataset of the same size could require a larger or smaller footprint. In contrast, the space required for STOMP is independent of both the structure of data and the subsequence length.

This severe memory requirement makes it impossible to compare the STOMP algorithm with Quick-Motif on the seismology data, since Quick-Motif often crashed with an out-of-memory error as we varied the value of m. However, we noticed that the memory footprint for Quick-Motif tends to be much smaller with smooth data. Therefore, instead of comparing performance of the algorithms on seismology data, in Table 3.7, we utilized the much smoother ECG dataset (used in [78]), which is an ideal dataset for both MK and Quick-Motif to achieve their best performance.

Table 3.7: Time required for motif discovery with $n = 2^{18}$, varying m, for various algorithms

	m			
Algorithm	512	1,024	2,048	4,096
STOMP	501s (14MB)	506s (14MB)	490s (14MB)	490s (14MB)
Quick-Motif	27s (65MB)	151s (90MB)	630s (295MB)	695s (101MB)
MK	2040s (1.1GB)	N/A (>2GB)	N/A (>2GB)	N/A (>2GB)

Clearly, both the runtime and memory requirement for STOMP are independent of the subsequence length. In contrast, Quick-Motif and MK both poorly scale in subsequence length in both runtime and memory usage. Note that the memory requirement of Quick-Motif is not monotonic in m, as reducing m from 4,096 to 2,048 requires three times as much memory. This is not a flaw in implementation (we used the author's own code), but a property of the algorithm itself.

As indicated in Figure 3.2, the Quick-Motif algorithm [55], the MK algorithm [65], and the original motif discovery by projection algorithm [25] can all be fast in the best

case. For example, if there happens to be a perfect (zero Euclidean distance) motif in the dataset, they will all discover it with $O(n)$ work (with high constants), and all algorithms can use this zero-valued best-so-far to prune all other possibilities for motif pairs. While we generally do not expect to have a zero-distance motif in real-valued data, a very close motif pair in a dataset with low intrinsic dimensionality (recall Figure 3.2) can offer similar speed ups. However, that describes the best case for all three algorithms. Consider instead the worst case (for example, the input signal is white noise, and all subsequences are effectively equidistant from each other), all three rival algorithms degenerate to $O(mn^2)$ (again, with high constants). In contrast, STOMP is unique in that its best case and worse case are identical, just $O(n^2)$. Because m can be as large as 2,000 (see Figure 3.13), this can produce a significant speedup. Moreover, as we will show in the next two sections, STOMP computes much more useful information than the two rival methods.

Before demonstrating this, we show that the experiments in the previous table were spurious for STOMP. We do not need to measure its time or memory footprint, because we can predict it precisely. To the best of our knowledge, this property is unique among all motif discovery algorithms proposed in the literature [25][55][65].

For STOMP (assuming only that $m \ll n$), given only n, we can predict how long the algorithm will take to terminate and how much memory it will consume, which is completely independent of the value of m and the data.

To do this, we need to do a single calibration run on the machine in question. With a time series of length n, we measure T, the time taken to compute the matrix profile,

and M, the (maximum) amount of memory consumed. Then, for any new length n_{new}, we can compute $T_{required}$, the time needed as the following:

$$T_{required} = \frac{T}{n^2} * n_{new}^2 \tag{3.6}$$

and we can compute $M_{required}$ the memory needed as the following:

$$M_{required} = \frac{M}{n} * n_{new} \tag{3.7}$$

As long as we avoid trivial cases, such as $m \sim n$, n_{new} is very small, or n is very small, and this formula will predict the resources needed with an error of less than 5%. To demonstrate this, we performed the following experiment. On our machine (a PC with Intel i7-2600@3.40GHz) we ran STOMP (Matlab version) on a random walk dataset of size 2^{18}, measuring the resources consumed. Then, as shown in Table 3.8, we use the formulas above to predict the resources needed to compute the Matrix Profile for datasets of size {2^{18}, 2^{19}, 2^{20}, 2^{21}}. Then we measured these values with actual experiments on random walk data. From Table 3.8, the agreement between our predictions and the observed values is clear.

Table 3.8: Time and memory required for STOMP, with m = 256, varying n

	STOMP runtime (memory)		
Input Size	Measured	Predicted	Relative Error
2^{18}	19.0 min (30MB)	19.0 min (30MB)	0% (0%)
2^{19}	75.6 min (60MB)	76.0 min (60MB)	0.5 % (0%)
2^{20}	313.2 min (121MB)	304 min (120MB)	3% (0.8%)
2^{21}	1252.8 min (242MB)	1216 min (240MB)	3% (0.8%)

This property has several desirable implications: we can carefully plan resources when performing analytics on large data archives; we can easily divide the work to parallel computing resources to finish our task in time; and we can show a perfectly accurate "progress bar" to a user who is using STOMP interactively.

3.3.4 Parameter Settings

As we have previously noted, STOMP (together with STAMP) is unique among motif discovery algorithms because it is parameter-free. In contrast, Random Projection [25] has four parameters, Quick-Motif [55] has three parameters, Tree-Motif has four parameters [104], MK [65] has one parameter, and FAST has three parameters [110].

That being said, the reader may wonder about the only input value besides the time series of interest: the subsequence length m. Note that this is also a required input for all the other existing techniques. However, we do not consider m to be a true parameter, as it is a user choice, reflecting her prior knowledge of the domain. Nevertheless, it is interesting to ask how sensitive motif discovery is to this choice; at least in the seismology domain that motivates us.

To test this, we edited the data above such that the two earthquakes in Figure 3.13.bottom happen exactly 13 minutes 20 seconds apart. We reran motif discovery with $m = 2,000$ (twenty seconds), with double that length ($m = 4,000$), and with half that length ($m = 1,000$). Figure 3.12 shows the result.

The results are reassuring. At least for earthquakes, motif discovery is not sensitive to the user input. Even a poor guess as to the best value for m, it will likely give accurate results.

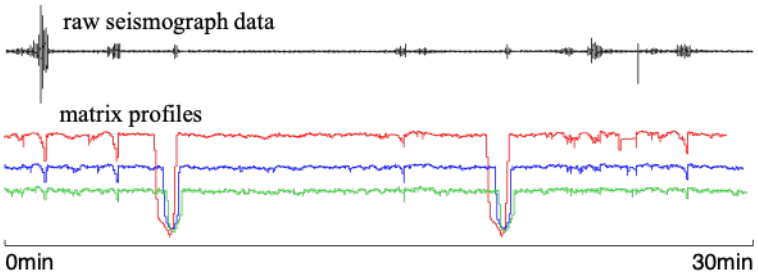

Figure 3.12: top) Thirty minutes of seismograph data that has the two earthquakes from Fig. 3.13.bottom occur at 6min-40s and 20min. bottom) The matrix profile computed if we use the suggested subsequence length 2,000 (blue), or if we use twice the length (red), or half that length (green).

3.3.5 Case Studies in Seismology: Infrequent Earthquake Case

To allow confirmation of the correctness and utility of STOMP, we begin by considering a dataset for which we know the result from external sources. On April 30th 1996, there was an earthquake of magnitude 2.12 in Sonoma County, California . Then on December 29th 2009, about 13.6 years later, there was another earthquake with a similar magnitude. We concatenated the two full days in question to create a single time series of length 17,279,800 (see Table 3.4 for timing results) and examined the top motifs with $m = 2,000$ (twenty seconds). Note that we are using the raw data as provided to us by the seismologists, we are not preprocessing it in anyway. As Figure 3.13.top illustrates, the top motif here is not an earthquake but an unusual sensor artifact [45].

There are a handful of other such artifacts; however, as shown in the bottom of Figure 3.13, the fifth best motif is the two occurrences of the earthquake. These misleading sensor artifacts are common, but they could be eliminated easily [45]. For example, the

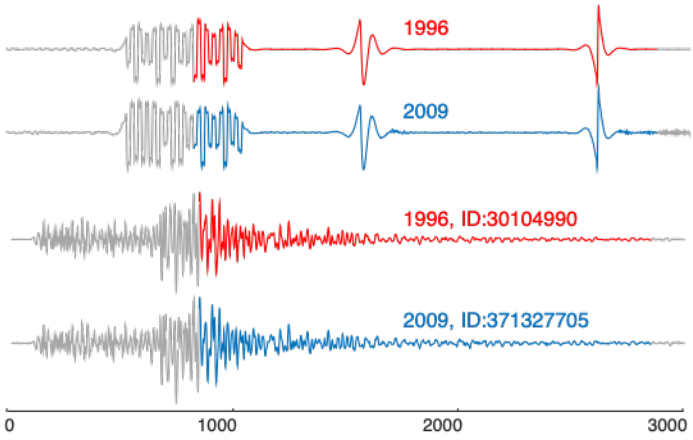

Figure 3.13: Motifs (colored) shown in context (gray). top) The top motif discovered in the Sonoma County dataset is a sensor artifact, as are the next three motifs (not shown). bottom) The fifth motif is two true occurrences of an earthquake that happen 4,992 days apart.

sensors could have a zero crossing rate that is an order of magnitude lower than true earthquakes.

3.3.6 Case Studies in Seismology: Earthquake Swarm Case

In the previous section, we discovered a repeating earthquake source that has a frequency of about once per 13.6 years. Here, we consider earthquakes that are tens of millions of times more frequent.

Forecasting volcanic eruptions is of critical importance in many parts of the world [97]. For example, on May 18th, 1980, Mount St. Helens had a paroxysmal eruption that killed 57 people [50]. It is conjectured that explosive eruptions are commonly preceded by elevated or accelerated gas emissions and seismicity; thus, seismology is a major tool for both monitoring and predicting such events. In Figure 3.14, we illustrate a short section of the matrix profile of a seismograph recording at Mount St Helens. It is important to

restate that this is not the raw seismograph data, but it is the matrix profile that STOMP computed from it.

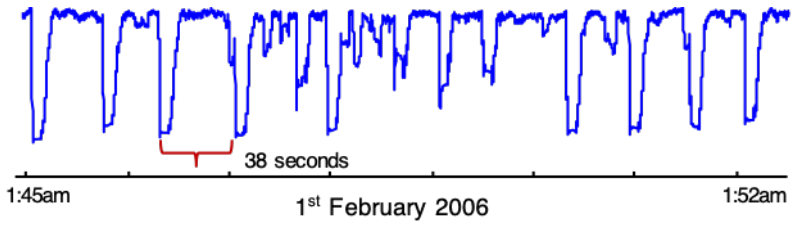

Figure 3.14: The matrix profile of a seven-minute snippet from a seismograph recording at Mount St Helens.

The image demonstrates a stunning regularity. Repeated earthquakes are occurring approximately once every thirty-eight seconds. This is consistent with the findings of a team from the US Geological Survey who reported that the earthquakes, which accompanied a dome-building eruption, appeared "... so regularly that we dubbed them 'drumbeats'. The period between successive drumbeats shifted slowly with time, but was 30–300 seconds." [50].

This example shows a significant advantage of our approach that we share with STAMP but no other motif discovery algorithm. Instead of computing only $O(1)$ distance values for the top k motifs, STOMP is computing all $O(n)$ distances from every subsequence to their nearest neighbors. By plotting the entire matrix profile, gain unexpected insights by viewing the motifs in context. For example, in the example above, we can see both the surprising periodicity of the earthquakes, and by comparing the smallest values in the matrix profile with the mean or maximum values, we can get a sense of how well the motifs are conserved relative to "chance" occurrences. It could also potentially indicate whether

there were changes to the earthquake source, reflecting changes in eruptive behavior over time.

A recent paper performed a similar analysis on the Mount Rainier volcano, making the interesting and unexpected discovery that the frequency of earthquakes is correlated with snowfall [5]. However, the paper bemoans at the number of ad-hoc "hacks" that needed to make such an exploration tenable. For example, "In order to save on computing time, we cut out detections that are unlikely to contain a repeating earthquake event by excluding events with a signal width," and "To save on computing time, we define that in order to be detected..." etc. [5]. However, the results in Table 3.4 indicate that we could bypass these issues by spending a few hours computing the full exact answers. This would eliminate the risk that some speedup "trick" erases an interesting and unexpected pattern.

3.3.7 Case Studies in Seismology: Detection of Repeated Low Frequency Earthquakes

In the previous sections, we showed how STOMP could help us detect repeating earthquake sources by evaluating the matrix profile of a single seismograph recording time series. Here we show that by providing the matrix profiles of multiple seismograph recording time series, STOMP allows us to detect low frequency earthquakes (LFEs). LFEs are of great importance to the seismology community, as they could "potentially contribute to seismic hazard forecasting by providing a new means to monitor slow slip at depth" [94]. LFEs recur episodically, often during bursts of tectonic 'tremor', which are considered superpositions of many LFEs in a short period of elevated seismic activity [93]. One traditional approach, known as 'matched filtering' identifies repeated LFEs by evaluating the cross-correlation between

continuous waveform data (time series) and a template waveform (subsequence) (e.g. [95]). However, this requires a suitable, carefully recorded template waveform of an LFE (an LFE subsequence) to have been identified in advance, which is very difficult or even impossible in many cases. In the face of this, similarity-join search through autocorrelation (e.g. [18]) has been used to detect LFEs in several studies. However, the traditional similarity-join search approach is computationally intensive (typically only one hour or less of continuously waveform data can be searched in feasible time), severely limiting the number and range of LFEs that can be detected.

Consider an example of LFE detection along the central San Andreas fault near Parkfield, CA. We search for LFEs in waveform data from a tremor burst that occurred on October, 6, 2007, in which many LFEs were detected by matched filtering [95]. As before, note that we are using the raw data as provided to us by the seismologists, we are not preprocessing it in anyway The LFE template (subsequence) in [95] was found by careful visual examination of seismic recording from multiple temporary seismic stations located close to the source (the green triangles in Figure 3.15; temporary stations were set up near a well-known earthquake source in this area), and subsequently also identified on more distant, permanent High Resolution Seismic Network (HRSN, the red triangles). Note that our task here is to detect all the LFEs automatically, and the only data available are those from the HRSN stations (the red triangles in Figure 3.15), since in most applications we do not know the earthquake source location (thus the data from the temporary stations) until well after the event.

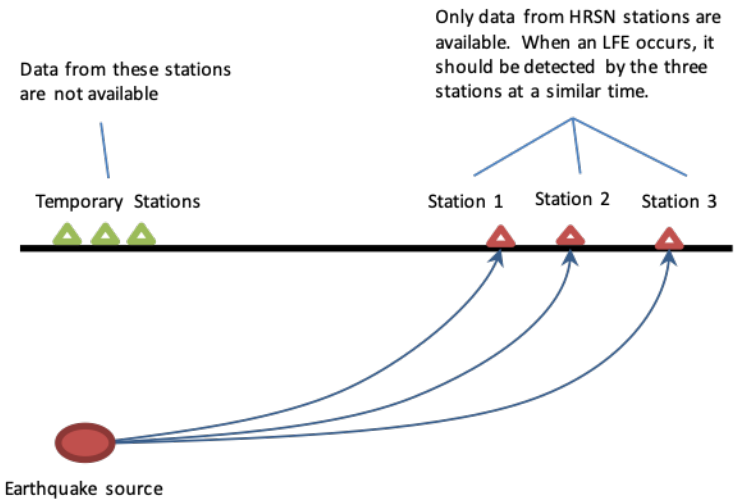

Figure 3.15: LFEs can be detected from the seismograph recording of HRSN stations.

Apart from the lack of the temporary station data, what makes our task even more difficult is that the data from HRSN stations are noisy and many contain a lot of false positives. For example, the top 15 motifs (repeating templates) found from the data of an HRSN station near central San Andreas fault are either sensor artifacts (similar to Figure 3.13) or instrument noise in the station itself. However, in spite of all these difficulties, we will demonstrate that STOMP allows us to detect LFEs from long seismic recordings.

We ran GPU-STOMPopt on the seismic recording time series from three HRSN stations for a 24-hour period spanning the tremor burst. The three HRSN stations are located close to each other. The data was sampled at 20Hz, for a total of ∼1.7 million samples per station time series. Figure 3.16 shows the sum of the three matrix profiles obtained.

The reader may wonder why we are summing the three matrix profiles here. This simple step greatly reduces the false positives in the data. As the three HRSN stations

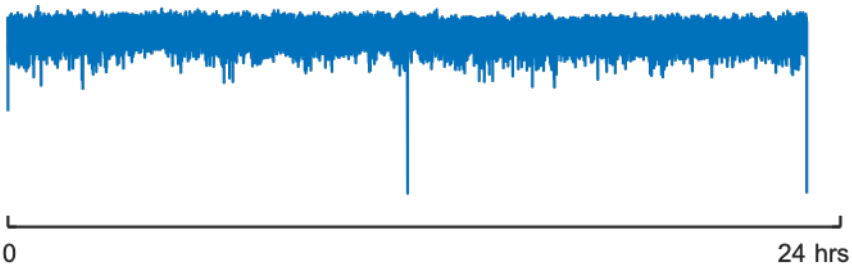

Figure 3.16: The sum of three matrix profiles of the 24-hour seismograph recording at three HRSN stations near the central San Andreas fault.

are located close to each other, when an LFE occurs, the stations should detect it at a similar time. As a result, the matrix profile values of the three stations should all be low at the occurrence of the LFE. The sum of the matrix profiles shows low values at such time instants, which strengthens the LFE signal and thus weakens the false positives, which are local to each sensor. We discovered that the top seven motifs identified in this way were either glitches in the waveform data (sensor artifacts, again, recall Figure 3.13), or signals that could not be separated into individual LFEs; however, as shown in Figure 3.17, the 8th best motif showed strong characteristics, in terms of frequency content, waveform shape and duration, of an LFE, and the origin time of this LFE is consistent with the results in [95], which may be regarded as the ground truth.

In contrast to [95], which detects the LFE pattern with weeks of enormous human effort, we are able to complete the same task automatically in approximately 3 minutes with GPU-STOMPopt on NVIDIA Tesla K80.

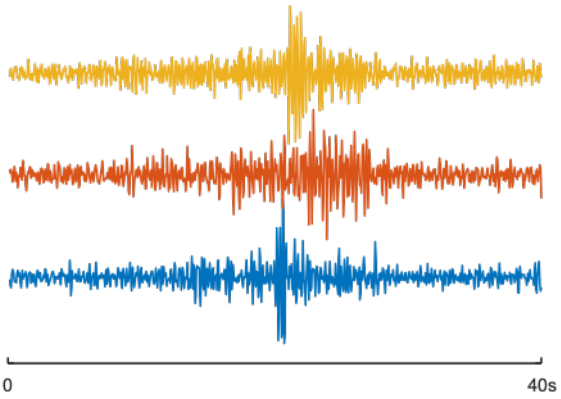

Figure 3.17: The 40-second LFE snippet detected from the three HRSN station time series.

3.3.8 A Case Study in Animal Behavior

While seismology is the primary motivator for this work, nothing about our algorithm assumes anything about the data's structure, or precludes us from considering other datasets. To demonstrate this, in this section, we briefly consider telemetry collected from Magellanic penguins (Spheniscus magellanicus). Adult Magellanic penguins can regularly dive to depths of between 20m to 50m deep in order to forage for prey, and may spend as long as fifteen minutes under water. The data was collected by attaching a small multi-channel data-logging device to the bird. The device recorded tri-axial acceleration, tri-axial magnetometry, pressure, etc. As shown in Figure 3.18, for simplicity we consider only Y-axis magnetometry. Note that, as with the seismology, we are not preprocessing this data source in in any way, no smoothing, not down sampling, etc.

An observer with binoculars labels the data; thus, we have a coarse ground truth for the animal's behavior. The full data consists of 1,048,575 data points recorded at 40 Hz (about 7.5 hours). We ran GPU-STOMPopt on this dataset, using a subsequence length of

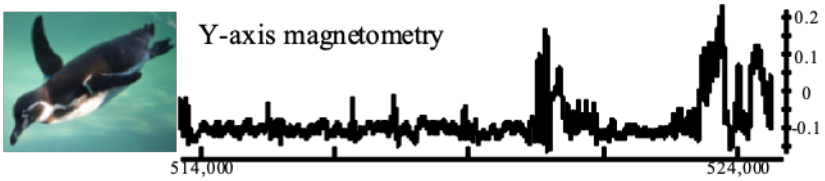

Figure 3.18: left) The Magellanic penguin is a strong swimmer. right) A four-minute snippet of the full dataset reveals high levels of noise and no obvious structure.

2,000. This took our algorithm just 49 seconds to compute. As shown in Figure 3.19, the top motif is a surprisingly well conserved "shark fin" like pattern.

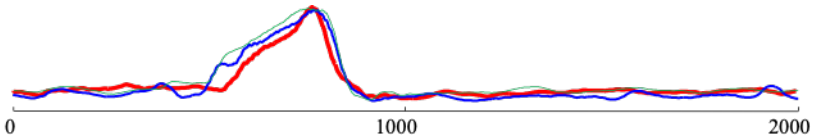

Figure 3.19: The top motif of length 2,000 discovered in the penguin dataset. Only three examples are shown for visual clarity, there are eight such patterns. This behavior may be part of a 'porpoise' maneuver.

What (if anything) does this pattern indicate? Suggestively, we observed this pattern does not occur in any of the regions labeled as nesting, walking, washing, etc., but only during regions labeled foraging. Could this motif be related to a diving (for food) behavior?

Fortunately, diving is the one behavior we can unambiguously determine from the data, as the pressure sensor reading increases by orders of magnitude when the penguin is under water. We discovered that the motif occurs moments before each dive, and nowhere else. This this pattern appears to be part of a ritual behavior made by the bird before diving. It has been reported that "The only time penguins are airborne is when they leap

out of the water. Penguins will often do this to get a gulp of air before diving back down for fish." Thus, we suspect this pattern in part of a 'porpoise' behavior [96].

Generally speaking, we see this example as typical of the interactions that motif discovery supports. In most cases, motif discovery is not the end of analyses, but only the beginning. By correlating the observed motifs with other (internal or external) data, we can form hypotheses and open avenues for further research. Recall the previous section; this is rather similar to the team studying Mount Rainier's seismology discovered that its earthquakes are correlated with snowfall [5]. We believe that the STOMP algorithm may enable many such unexpected discoveries in a vast array of domains.

3.4 Conclusion

In this chapter we introduced STOMP, a new algorithm for time series motif discovery, and showed that it is theoretically and empirically faster than its strongest rivals in the literature, STAMP [109], Quick-Motif [55] and MK [65]. In the limited domain of seismology, we showed that STOMP is at least as fast as the recently introduced FAST algorithm [110], but STOMP does not allow false negatives and does not need careful parameter tuning. Moreover, for datasets and subsequences lengths encountered in the real world, STOMP requires one to three orders of magnitude less memory than rival methods. Thus, even if we are willing to wait a longer period of time for the rival methods to search a large (ten million-plus) dataset, we will almost certainly run out of main memory. Given that these algorithms require random access to the data, disk-based implementations are infeasible. This is not a gap that is likely to be closed by a new implementation of these algorithms,

because STOMP is unique among motif discovery algorithms in not extracting subsequences, but performing all the computations in-situ.

We further demonstrated optimizations that allow STOMP to take advantage of GPU architecture, opening an even greater performance gap and allowing the first exact motif search in a time series of length one hundred and forty-three-million.

Chapter 4

SCAMP - A Distributed, Scalable, Matrix Profile Framework

4.1 Introduction and Background

In this chapter, we show that with several novel insights we can push the motif discovery envelope even further than with GPU-STOMPopt using a scalable framework in conjunction with a deployment to commercial GPU clusters in the cloud. We demonstrate the utility of our ideas with detailed case studies in seismology, demonstrating that the efficiency of our algorithm allows us to exhaustively consider datasets that are currently only approximately searchable, allowing us to find subtle precursor earthquakes that had previously escaped attention, and other novel seismic regularities. To meet the needs of domain experts, we present a cloud-scale framework called SCAMP (SCAlable Matrix Profile) that expands the purview of exact motif discovery. We summarize our major contributions below:

1. We provide a general distributed framework for the ultra-scalable computation of the Matrix Profile [109]. Both the performance and numerical stability are greatly improved via our method when dealing with long time series.

2. Our framework allows us to work with time series data which do not fit wholly into GPU memory, allowing MPs to be computed which are larger than previously considered.

3. We introduce novel numerical methods to increase performance and improve stability of the MP computation; this allows the use of single-precision floating-point calculations for some datasets, which allows our methods to be applied to larger datasets at a cheaper amortized cost.

4. We deployed a fault-tolerant framework that is compatible with "spot" instances [7], which major cloud providers (Amazon, Google, and Microsoft) offer at a major discount, making motif discovery more affordable.

5. We provide a freely available open-source implementation of our framework which runs on Amazon Web Services in a cluster of instances equipped with Nvidia Tesla V100 GPUs, as well as optimized CPU code at [114].

Much of the terminology for this chapter remains the same as in Chapter 3, but there are a few additional considerations to be wary of.

4.1.1 Observations on Precision

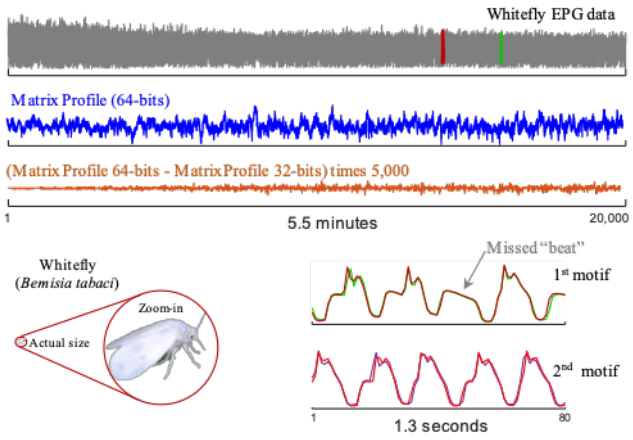

Figure 4.1: top-row) A snippet of whitefly insect EPG data. second-row) The MP computed with 64-bit precision. third-row) Because the 64-bit and 32-bit MPs are visually identical at this scale, we subtracted them, and multiplied the difference by 5,000. bottom-row) The whitefly is tiny, yet it produces well conserved motifs.

Several independent research groups have noted that for some time series retrieval tasks, 64-bit precision is unnecessarily precise [12][103]. Researchers have shown that reduced precision can be exploited to have significant performance benefits with minimal observable difference in quality of results [103][40]. This observation has been heavily exploited in deep learning [43][40]; however, it is rarely exploited for time series, except for allowing the use of Minimal Description Length to score and rank models [12], which is orthogonal to scalability considerations. Figure 4.1 shows an MP computed on some insect electrical penetration graph (EPG) data using 64-bit precision.

This plot suggests that the difference between MPs computed at 64 and 32-bit precision is so small it does not affect the motifs discovered, and is not visible unless we multiply the difference by a large constant; however, we must consider two caveats:

- The time series shown in Figure 4.1 is relatively short. To address ever longer time series, there is more potential for accumulated floating-point error to impact the result [47]. Even in this example we can see that the difference vector gets larger as we scan from left to right (Figure 4.1.third-row). We address this issue with our tiling scheme in 4.2.1.

- The information contained in the time series in Figure 4.1 is contained within a small range. This is true for some types of data, such as ECGs, accelerometer and gyroscope readings; however, there are also a handful of domains for which this is not true, such as seismology. A "great" earthquake has a magnitude of 8 or greater, but humans can feel earthquakes with magnitudes as low as 2.5, a difference of more than five orders of magnitude. Processing raw data with a large dynamic range is non-trivial (see Sections 4.2.1, 4.2.6, and 4.3.4).

Before proceeding, we note that this illustration offers another example of the utility of motif discovery. The time series in Figure 4.1 is a fraction of an entomologist's data archive [59]. The 2nd motif represents ingestion of xylem sap behavior [86], which is common and immediately recognizable by an entomologist; however, the 1st motif was unexpected: there is a "missed beat" during the xylem sap ingestion cycle. If we had observed a single example, we could attribute it to chance or noise; however, motif discovery shows us that there are at least two strongly conserved examples. This suggests that there exists some semantic meaning to this motif, which entomologists are currently exploring [59].

4.2 The SCAMP Framework

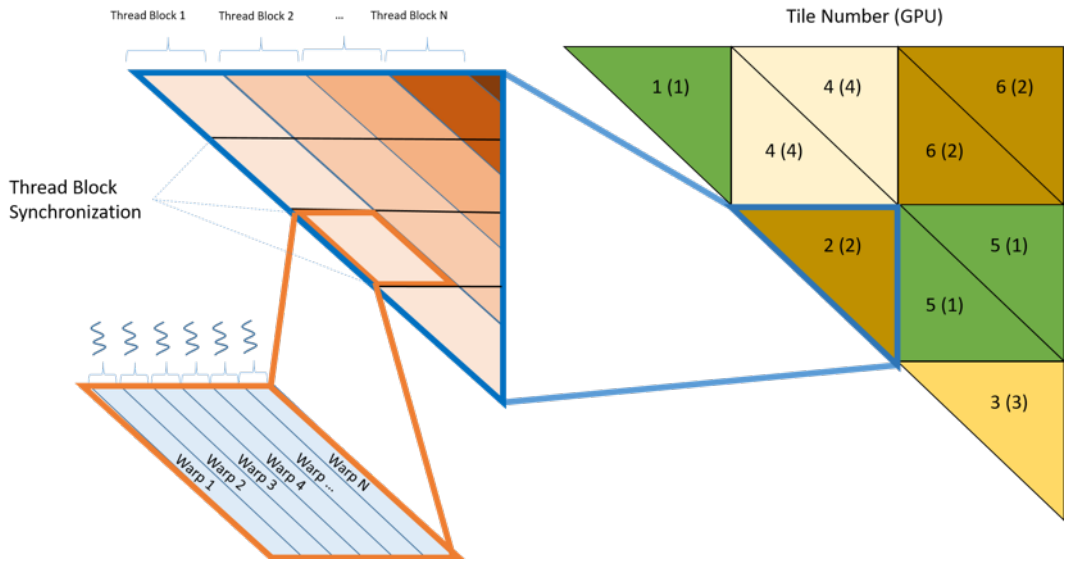

Figure 4.2: left) The GPU-STOMPOPT execution pattern, which is shared with the SCAMPTILE algorithm. right) The SCAMP tiling scheme using 4 GPUs. The illustration of the tiling scheme is for self-joins only; the lower triangular tile is computed with the same implementation, but with the inputs transposed.

To compute large MPs, we introduce the SCAMP framework that can be used by a cluster with a host and one or more workers. A host can be a local machine, or a master server. A worker can be a CPU-based system or an accelerator (e.g., a GPU), following the host's direction. A cluster refers to the combination of a host and all of its associated workers. This can be the typical group of co-located nodes in a cloud, or a single node with accelerators attached (e.g. a server equipped with several GPUs). Additionally, SCAMP improves several aspects of GPU-STOMPopt, yielding a several-fold improvement in performance and allows efficient exploitation of newer GPU hardware. We explain these improvements in detail in the following sections.

4.2.1 Tiling Scheme

Rather than computing the entire distance matrix in one operation, we split it into tiles. Each tile independently computes an AB-join between two segments of the input time series. This allows the computation to scale to very large input sizes and distribute the work to many independent machines, as depicted in Figure 4.2.right. The host maintains information about its workers, such as the number and type of available GPUs, the memory capacity, and the CPU speed, to determine a tile width that can saturate its workers. The host generates tiles of this width and delegates them among the workers. For simplicity, this paper assumes that all of the workers are homogeneous (V100 GPUs) and that the most effective tile size ($\sim$1 million) fully saturates each worker during execution. This tile width is currently discovered empirically, but could be hard-coded once it is known a given system configuration.

4.2.2 Host Algorithm

Table 4.1: The $SCAMP_{HOST}$ Algorithm.

Procedure $SCAMP_{HOST}$
Require: T a time series, window length m, and tile size s
1: tiling := GetTiling($len(T)$, s)
2: stats := PrecomputeStats(T,m)
3: **for** row, col **in** tiling **do**
4: tile := CreateTile(T, m, stats, row, col, s)
5: globalWorkQueue.add(tile)
6: **end for**
7: StartAsynchronousWorkers()
8: P, I := WaitForWorkerResults()
9: **return** P, I

The host executes the $SCAMP_{HOST}$ algorithm, which employs multiple asynchronous workers, which could be threads or other nodes in a cluster (see Table 4.1). Line 1 determines the appropriate tiling for the problem instance and the relative tile execution order. Line 2 precomputes all necessary statistics of T needed to compute distances between subsequences. Lines 3-5 initialize a data structure containing all information necessary to compute the result for each tile in our problem instance, and insert the tile into a global work queue. Line 7, initializes asynchronous workers, who extract work from the queue. Line 8 retrieves and merges and the tile result and Line 9 outputs the result.

4.2.3 Tile Computation

Table 4.2: SCAMP Tile Computation

Procedure $SCAMP_{TILE}$
Require: $workQueueOfTiles$ globalWorkQueue
1: **while** globalWorkQueue **not EMPTY do**
2: tile := globalWorkQueue.GetItem()
3: **if** tile **is null then**
4: **return**
5: **end if**
6: A := tile.A, B := tile.B
7: mp := tile.mp, mpi := tile.mpi, stats := tile.stats
8: QT := SlidingDotProducts(A,B)
9: mp, mpi := DoTriangularTile(A, B, stats, QT, mp, mpi)
10: QT := SlidingDotProducts (B,A)
11: mp, mpi := DoTriangularTile (B,A, stats, QT, mp, mpi)
12: ReturnTileToHost(mp, mpi)
13: **end while**
14: **return**

Workers execute the $SCAMP_{TILE}$ algorithm to compute each tile's intermediate result (see Table 4.2), while unprocessed tiles remain in the global work queue. Line 2-7 extracts a tile from the work queue, along with its relevant information from the tile

structure. Line 8 computes initial dot product values associated with the upper triangular tile. Line 9 executes an architecture-optimized kernel to compute the local MP and Index for that tile. Lines 10 and 11 compute the initial dot product values associated with the lower triangular tile and the result associated with that tile. The tile's computation similar to GPU-STOMPopt from Chapter 3, with additional optimizations, described in the following sections.

4.2.4 Optimizations

The host may run out of memory if tiles are sufficiently small and too many are pre-allocated; however, this can be overcome via optimization. For example, in a single node deployment, each worker, rather than the host, can construct the full tile upon its execution. In a distributed deployment, the maximum number of tiles in the queue can be limited, and more work can be added as each tile's processing completes. Further, it is possible to cache the best-so-far MP values as tiles computed by workers, enabling subsequent tiles to be initialized with more up-do-date MP values. These optimizations reduce the number of memory accesses during computation, but have been omitted from Tables 4.1 and 4.2 for simplicity of presentation.

Chapter 3 established that there is symmetry in the distance matrix for self-joins; here, we note that the memory access pattern and the order of distance computations in SCAMP and GPU-STOMPopt are similarly symmetric. The lower-triangular portion of the distance matrix Figure (3.5) can be computed using the same subroutine as the upper-triangular portion simply by transposing the input. The SCAMP framework exploits this property to implement joins.

4.2.5 Comparison to GPU-STOMPopt

Beyond the scope of the preceding discussion, SCAMP offers several distinct advantages over GPU-STOMPopt:

- **Extensibility**: Since tiles are computed independently, SCAMP can provide different options for each tile's computation, which offers a pathway to run SCAMP on a heterogeneous compute infrastructure. See Chapter 6 for additional information on how SCAMP can be used to perform other kinds of computation besides generating the Matrix Profile

- **Numerical Stability**: Each new tile introduces a 'reset' point for SCAMP's extrapolation. When a new tile computation begins, SCAMP directly computes high-precision initial dot products of the distance matrix at that row and column. This reduces the likelihood that rounding errors propagate along diagonals. In contrast, GPU-STOMPopt extrapolates the diagonals of the distance matrix from a single initial value.

- **Fault-Tolerance**: $SCAMP_{TILE}$ independently issues and completes processing for each tile; as a result, it is inherently preemptable, which increases the fault-tolerance of our framework. If a worker executing a tile "dies" or otherwise fails to complete its work, the host can simply reissue a new instance of the incomplete tile into the work queue. As mentioned in Section 4.1 , many commercial cloud providers allow users to purchase spot instances at discounted prices. Spot instances are only useable by fault-tolerant applications because the cloud provider can kill the instance at any

time. Thus, SCAMP provides a pathway for lower-cost cloud-based MP computation, which GPU-STOMPopt cannot provide. SCAMP users can increase the number of compute resources purchased at a fixed cost point, which increases the size of the time series datasets they can process using SCAMP.

4.2.6 Numerical Optimization and Unrolling

To improve performance and numerical stability, SCAMP reorders GPU-STOMPopt's floating point computations and replaces its sliding dot product update (Equation 3.2 and 3.5) with a centered-sum-of-products formula (Equations 4.1 to 4.5).

These transformations reduce each thread's demand for shared memory; at the same time, increasing the amount of shared memory allocated to each thread, allows each worker to compute four separate diagonals (Figure 4.3).

$$df_0 = 0; df_i = \frac{T_{i+m-1} - T_{i-1}}{2} \tag{4.1}$$

$$dg_0 = 0; dg_i = (T_{i+m-1} - \mu_i) + (T_{i-1} - \mu_{i-1}) \tag{4.2}$$

$$\overline{QT}_{i,j} = \overline{QT}_{i-1,j-1} + df_i dg_j + dg_j dg_i \tag{4.3}$$

$$P_{i,j} = \overline{QT}_{i,j} * \frac{1}{||T_{i,m} - \mu_i||} * \frac{1}{||T_{j,m} - \mu_j||} \tag{4.4}$$

$$D_{i,j} = \sqrt{2m(1 - P_{i,j})} \tag{4.5}$$

Equations 4.1 and 4.2 precompute the terms used in the sum-of-products update formula of Equation 4.3, and incorporate incremental mean centering into the update.

Equations 4.1, 4.2, and 4.3 are specific to self-joins and are a special case of a more general formula for an AB-join [114]. This new formula reduces the number of incorrectly rounded bits.

Equation 4.4 replaces the Euclidean distance used in previous MP definitions ([109] and Chapter 3) with the Pearson Correlation; Pearson Correlation can be computed incrementally using fewer computations than ED, and can be converted to z-normalized ED in $O(1)$ by Equation 4.5. SCAMP also precomputes the inverse L_2-norms in Equation 4.4 to eliminate redundant division operations from SCAMP's inner loop.

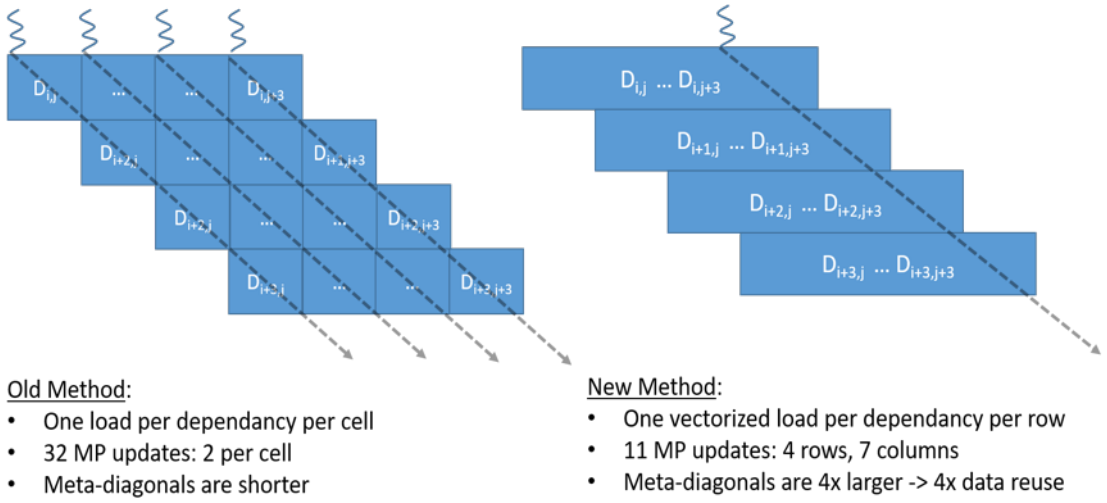

Figure 4.3: One iteration of the innermost loop of GPU-STOMPopt (left) and SCAMP (right). Self-joins require only half of the distance matrix, but we must track both the MP value for the columns and for the rows. AB-joins only require the columns or the rows.

Unrolling the innermost loop 4x requires each thread to compute 16 new distances per iteration (four distances for each of four diagonals), while ensuring the per-thread-block register and memory usage remains low enough to achieve 50% occupancy on a Tesla V100 GPU (see Ref. [70] for details). MP computation on the GPU is bound by shared memory

loads not compute time. Unrolling permits SCAMP to use vectorized shared memory loads for dependencies, enabling consolidation of shared memory transactions.

SCAMP tracks the maximum per-row and per-column distances and updates the corresponding MP value in shared memory when an improvement occurs, resulting in a single update per row. In contrast, GPU-STOMPopt compares every newly computed distance to the MP cache.

4.2.7 Floating-point Precision Options

We evaluated SCAMP under two precision modes:

$SCAMP_{DP}$ performs all computation and stores all intermediate shared memory values in double-precision. $SCAMP_{DP}$ generated accurate results for all datasets that we tested, regardless of size, noise, ill-conditioned regions, etc.

$SCAMP_{SP}$ performs all computation and stores all intermediate shared memory values in single-precision, which increasing performance and memory utilization by $\sim$2x. $SCAMP_{SP}$ was adequate for highly regular datasets, such as ECG or accelerometer data, but may yield incorrect results for ill-conditioned data (see Section 4.3.4 for a detailed analysis). Using vectorized shared memory loads, $SCAMP_{SP}$ executes two 128-bit loads per column dependency and one 128-bit load per row dependency. This enabled all intermediate values to be stored in registers without spilling.

We tested SCAMP using half-precision (16-bit) floating-point operations but found that SCAMP identified incorrect motifs for many data sets; we do not consider half-precision any further.

4.2.8 Multi-Node AWS Deployment

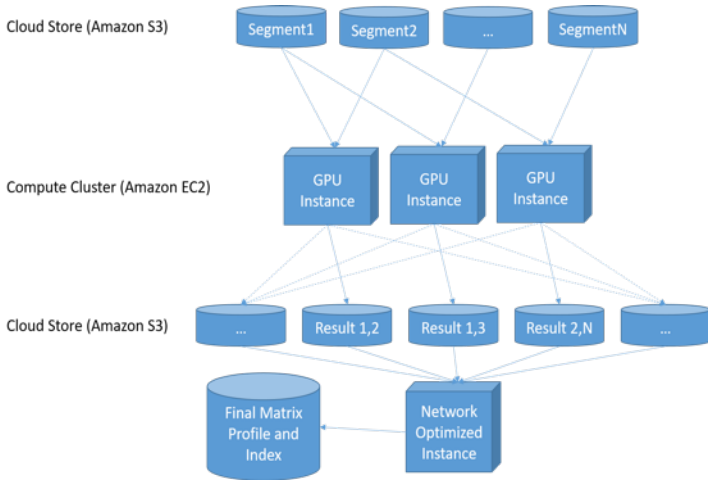

Figure 4.4: Illustration of how to distribute SCAMP in a cluster of GPU instances on AWS.

We deployed SCAMP on Amazon Web Services (AWS), as representative commercially available cloud platform (see Figure 4.4). We first partition our time series data set into equal-sized chunks ranging from 20 to 100 million elements. There is a tradeoff here between the overhead of initiating new jobs, intermediate data size, and the risk of a job being preempted and losing work. We compress each chunk and store it on the cloud (Amazon S3), where it can be read by worker nodes. There is existing work on array stores, [74], that might be leveraged in providing access to the input array among worker nodes, but for simplicity we defer a study on these methods to future work.

We use AWS batch to set up a job queue backed by a compute cluster of p3.16xlarge spot instances. We issue an array batch job in which each job computes the MP for one tile. We issue one job per worker, and the tile size is specified to ensure full saturation of each

worker's compute resources. This maximizes throughput of the processing pipeline without risking exorbitant progress loss if Amazon preempts a worker.

Each worker first copies and decompresses its input segments corresponding to the row and column of its tile. Each tile has two inputs: a segment corresponding to the tile-row, and another corresponding to the tile-column; each job computes an AB-join on the inputs. Next, the worker executes $SCAMP_{HOST}$ on the input, further subdividing the tile among its GPUs. Once the worker computes the MP and index associated with the tile, the result is compressed and written back to Amazon S3.

Each job dequeues after it terminates. After all jobs terminate, another job decompresses and merges each tile's MP into the final result; as long as intermediate data growth is limited, this is relatively simple. In a 1 billion datapoint experiment, we merged 196 GB of intermediate results in $\sim$1 hour using one AWS machine. The merging step could be further parallelized using a framework such as MapReduce [30].

Intermediate output data volumes can grow to tens or hundreds of gigabytes for input sizes up to 1 billion elements. Small tile sizes produce too much local information to reasonably store. SCAMP's space requirement is $O(RN)$ where R is the number of tile rows, and N is the length of the final MP. If the tile size is 1, then $R = N$ and processing one billion elements necessitates storing the distance matrix ($\sim$1 quintillion values). If each intermediate value is eight bytes compressed on disk, the total storage requirement would be $\sim$8 exabytes, the estimated aggregate storage capacity of Google's datacenters in 2014 [66].

4.3 Emperical Evaluation

All experiments reported here are reproducible. All code and data (and additional experiments omitted for brevity) are archived in perpetuity [114].

Table 4.3 reports the result of a direct comparison of SCAMP to GPU-STOMPopt using random walk datasets of various lengths. The first column reports the performance of GPU-STOMPopt using the code from STOMP 3 on an Nvidia Tesla K80 GPU. The results here are similar, but vary slightly due to a change in the timing of the experiment to improve precision.

Table 4.3: SCAMP Runtime Evaluation on Various Architectures

Algorithm	**STOMP-GPUopt**		**SCAMP**	
Architecture	K80	V100	V100	V100
Precision	DP	DP	DP	SP
2^{18}	3.04s	0.34s (**8.9x**)	0.28s (**10.9x**)	0.24s (**12.7x**)
2^{19}	11.4s	1.24s (**9.2x**)	0.68s (**16.8x**)	0.57s (**20.1x**)
2^{20}	44.1s	4.81s (**9.2x**)	2.05s (**21.5x**)	1.42s (**31.1x**)
2^{21}	174s	19.0s (**9.2x**)	6.99s (**24.9x**)	4.38s (**39.8x**)
2^{22}	629s	69.2s (**9.1x**)	25.8s (**24.4x**)	15.5s (**40.7x**)
2^{23}	2514s	277s (**9.1x**)	96.8s (**26.0x**)	52.5s (**47.9x**)

The second column reports the execution time of the same code (still GPU-STOMPopt) running on a single Nvidia Tesla V100 SXM2 on Amazon EC2. The reported speedup is due to the V100's higher instruction throughput compared to the K80, which is bottlenecked by the latency of atomic updates to shared memory. Nvidia implemented shared memory atomics in hardware and included them in their instruction set architecture (ISA) starting with the Maxwell GPU family [26]; they are no longer a performance bottleneck on newer GPU architectures. The third and fourth columns report the execution

time and speedups (relative to Column 1) of $SCAMP_{DP}$ and $SCAMP_{SP}$ running on the V100 GPU. The reported speedups are due to the optimizations described in Sections 4.2.1, 4.2.6, and 4.2.7. $SCAMP_{SP}$ does not always produce the same result as $SCAMP_{DP}$ due to the difference in computational precision.

4.3.1 Scalability

Figure 4.5 depicts an analytical performance model for SCAMP's execution time under ideal conditions. Given the runtime of SCAMP (T_o) on one GPU on a dataset of a size (N_o) which sufficiently saturates compute performance, we construct an analytical model (Equation 4.6) to estimate SCAMP's execution time across G GPUs on a time series of length N under ideal assumptions (e.g., no communication overhead).

$$N = N_o\sqrt{\frac{TG}{T_o}} \tag{4.6}$$

N_o and T_o are initialization parameters provided by one trial run on a single V100 GPU. We use this equation and the $SCAMP_{DP}$ runtime for input size 2^{23} (Table 4.3) to construct the model in Figure 4.5.

Each data point in Figure 4.5 corresponds to an experiment we ran, which demonstrates that the empirical model is highly accurate. The data for our distributed workloads in the next section also align well was this plot but were not included due readability constraints. More detail is available on our supporting webpage [114]. Under this model, the cost of a problem remains constant if there is no distributed overhead. For example, to compute a join of 530 million using double-precision, one can either use 8 GPUs for 8 hours, or

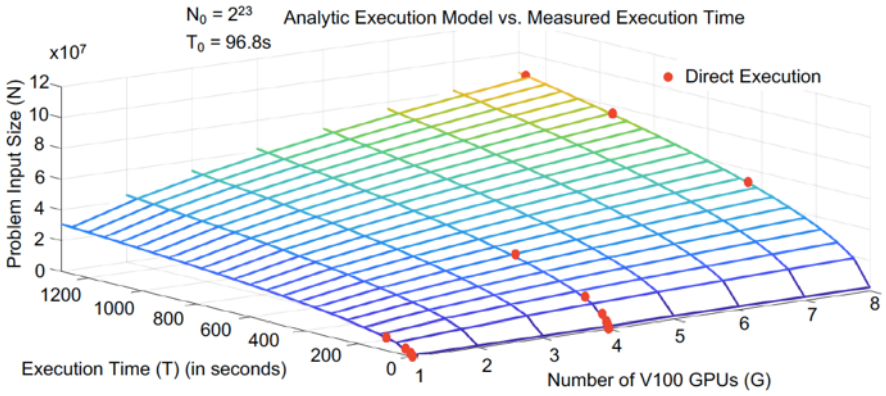

Figure 4.5: Equation 4.6 plotted using N_o and T_o from Table 4.3, the V100 double precision result for a dataset with 2^{23} data points. Dots correspond to values measured during experiments reported in this paper. Results are for a single non-preemptable instance equipped with G GPUs. Equation 4.6 also generalizes to multi-instance distributed workloads.

64 GPUs for 1 hour. The cost is identical as long as there is no difference in the cost per hour for GPU compute time.

4.3.2 Distributed Performance: p3 spot instances

Next, we evaluate SCAMP's performance on two very large earthquake datasets. Both experiments ran on 40 V100 GPUs, each in a different configuration, on an AWS EC2 spot instance fleet. A spot instance fleet automatically provisions a consistent number of spot instances for the job queue. If one instance is preempted, AWS provisions another for the fleet as long as there are available instances. A spot instance user accesses compute resources not sold to customers who pay full price for non-preemptable instances. Spot instance prices increase when demand is high; when demand is low, the provider loses money, but mitigates losses by selling preemptable access to the highest bidder.

The Parkfield dataset ran on a five p3.16xlarge spot instance fleet, where each instance is equipped with eight V100 GPUs. The p3.16xlarge instances were in high demand at the time of the experiment: many jobs remained queued at times that AWS could not provide capacity to execute; we were only charged for active GPU compute time. The Cascadia Subduction Zone dataset ran on ten Amazon EC2 p3.8xlarge instances each equipped with four V100 GPUs. These instances were in lower demand than those used for the Parkfield data set experiments, allowing faster job completion time with less queuing overhead. The spot price of Amazon spot instances is dynamic and demand-driven [7], and we were charged a higher spot price. Table 4.4 reports the results of these experiments.

Table 4.4: Summary of various distributed runs on AWS spot instances

Dataset	**Parkfield**	**Cascadia**
Size	1 Billion	1 Billion
Tile Size	∼52M (1 month)	∼ 25M (2 weeks)
Total GPU time	375.2 hours	375.3 hours
Spot Job Time	2.5 days	10hours 3min
Approximate Spot Cost	480 USD	620 USD
Intermediate Data Size	102.2 GB	196.4 GB

Table 4.5: Optimized CPU and GPU $SCAMP_{DP}$ cost on a single AWS instance

Instance Type **Hardware** **Cost/hr**	c5.18xlarge 72 Cores 3.06 USD/hr	p3.2xlargs 1 Tesla V100 3.06 USD/hr
Input Size	Runtime (s)	Runtime (s) (**speedup**)
2^{18}	7	0.28 (**25x**)
2^{19}	14	0.68 (**20x**)
2^{20}	32	2.0 (**16x**)
2^{21}	76	7.0 (**11x**)
2^{22}	252	25.8 (**9.8x**)
2^{23}	933	96.8 (**9.6x**)

4.3.3 CPU Comparison

Table 4.5 compares the performance of our GPU implementation of SCAMPDP to a CPU implementation running on a 72-core c5.18xlarge instance (Intel Skylake CPU). The CPU implementation saturates performance at an input size of 2^{21}, after which its runtime scales quadratically, as expected. At the time of writing, the c5.18xlarge has the same on-demand price on AWS as a p3.2xlarge which employs one V100 GPU. While it is difficult to compare cross-architecture performance, we can and do compare price per performance, which is shown in bold as a factor of improvement of the GPU over the CPU. In this case, the GPU is approximately one order of magnitude more cost-efficient. The price per performance for smaller input sizes is an imperfect basis for comparison: we could have used a smaller spot instance type to achieve better price per performance on a CPU when small input data sizes fail to saturate the 72 available cores on the c5.18xlarge instance.

4.3.4 Precision Evaluation

Consider the three data snippets shown in Figure 4.6. Each has a constant region longer than the chosen motif length m. Constant regions are a source of numerical instability. Many scientists are interested in the similarity of z-normalized subsequences. Z-normalization divides each data point by the standard deviation of the entire subsequence. For a constant region, the standard deviation is 0. Near-constant subsequences are also problematic, because they pass a bit-level test for two distinct values but result in division by a number very close to 0.

Constant regions are common. For example, in medical datasets, we have observed constant regions caused by:

Disconnection Artifacts: These may occur due to disconnection of a monitoring lead, e.g., during a bed change.

Hard-Limit Artifacts: Some devices have a minimum and/or maximum threshold defined by a physical limit of the technology. If the true value exceeds the limit for a period of time, a constant value occurs for the duration (Figure 4.6.center).

Low Precision Artifacts: Many devices record at low-precision fixed-point; observed constant values may not be constant at a higher precision.

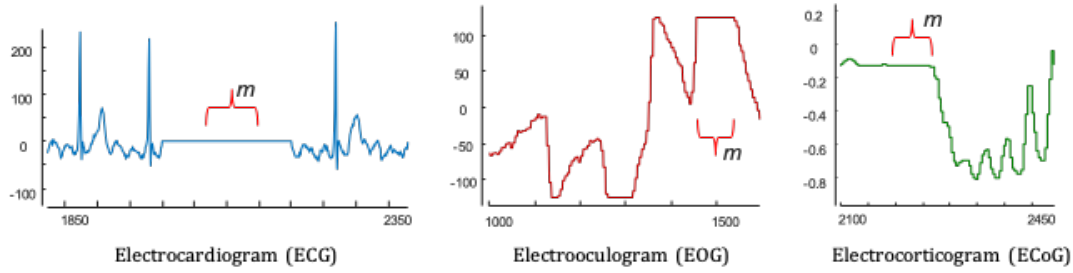

Figure 4.6: Three time series containing a constant region caused by different issue [29]. left) An ECG (heart) with a disconnection artifact. center) An EOG (eye movement) with a hard-limit artifact. right) An ECoG (finger flexion) with constant region caused by low precision recording.

In most cases, disconnection artifacts saturate to a Pearson Correlation of 1 or a z-normalized Euclidean Distance of 0, and are removed later via a post processing step. If small peaks and valleys are important in a low-precision artifact scenario, the MP can be computed and stored in double-precision.

4.3.5 Comparison with Previous Update Method

Figure 4.7 compares SCAMP's update method (Equations 4.1-4.5) with the prior method implemented in GPU-STOMPopt (Equations 3.2 and 3.5). We compute the result first in double precision, then plot the absolute error in computed Pearson Correlation between the double and single precision for both SCAMP and GPU-STOMPopt.

The bottom and middle of Figure 4.7 elucidate how Equations 3.2 and 3.5 (GPU-STOMPopt's update method), completely fail in single precision on this dataset. We capped the error at 1 for GPU-STOMPopt, which is half of the range of Pearson Correlation. The actual values reported by GPU-STOMPopt were many times larger than the entire range of Pearson Correlation.

In contrast, SCAMP only exhibits error in constant regions that arise due to disconnection artifacts. Here, a domain expert can easily clean up SCAMP's results with minimal effort by omitting these regions from consideration when analyzing the output of SCAMP. In contrast, GPU-STOMPopt fails to produce a meaningful result across almost most of the dataset.

4.3.6 General Considerations for Precision

Next, we analyze the effect of reducing precision on various datasets of different lengths. We use a tile size of 1 million for SCAMP while GPU-STOMPopt computes across the entire length of the input in one go, as it does not perform tiling. We generate the MP using $SCAMP_{DP}$, $SCAMP_{SP}$ and GPU-STOMPopt with single and double precision. We used a window length longer than the longest flat artifact region in the data, to allow us

to isolate errors caused by the update formula from the inherent loss of information from artifacts that cannot be represented in lower precision.

Table 4.6 presents the results of the experiment. Altogether SCAMP was three or more orders of magnitude more accurate than STOMP on these datasets. Each entry in Table 4.6 is the maximum absolute error found between the double and single-precision MP calculations. We highlight absolute errors that exceed 0.01 in red to emphasize that a domain scientist would not consider these results sufficiently accurate to use or report.

$SCAMP_{SP}$ suffers substantial accuracy loss compared to $SCAMP_{DP}$ but achieves higher performance. If a user's dataset and application can tolerate the loss of accuracy, there is much to be gained in terms of efficiency. We observe that $SCAMP_{SP}$ works well on data that is highly regular with a small min-max range, exemplified by ECG data.

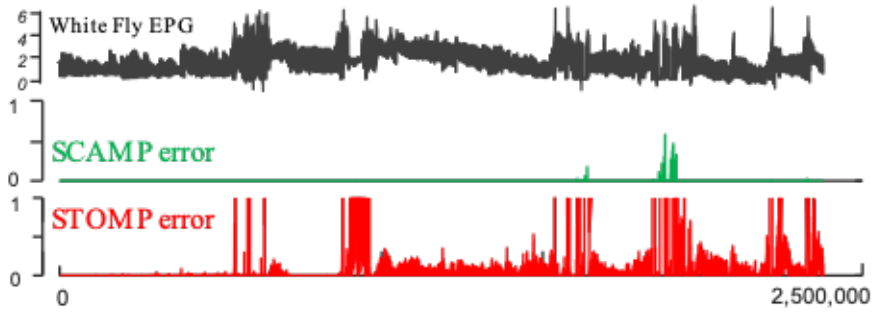

Figure 4.7: Single precision error comparison between GPU-STOMPOPT and SCAMP on White Fly EPG dataset. top) original data. middle) SCAMP absolute error. bottom) GPU-STOMP absolute error.

$SCAMP_{SP}$ completely fails on the Earthquake dataset in Table 6. This is because the large earthquake's signal has a magnitude greater than 10^7, which cannot be represented

Table 4.6: Absolute error (Pearson Correlation) for various datasets/algorithms. Red denotes high error

Maximum Absolute Error	Size (m)	$SCAMP_{SP}$	$STOMP_{SP}$
Whitefly EPG	2.5M (1000)	$3.75 * 10^{-2}$	$1.89 * 10^{1}$
ECG	8.4M (100)	$3.14 * 10^{-4}$	$2.07 * 10^{-3}$
Earthquake	1.7M (200)	$6.35 * 10^{-1}$	$3.17 * 10^{3}$
Power Demand	10M (4000)	$4.85 * 10^{-2}$	$2.22 * 10^{-1}$
Chicken	9M (1000)	$4.92 * 10^{-2}$	$2.27 * 10^{1}$
99.9 percentile absolute error	Size (m)	$SCAMP_{SP}$	$STOMP_{SP}$
Whitefly EPG	2.5M (1000)	$3.00 * 10^{-3}$	$1.55 * 10^{1}$
ECG	8.4M (100)	$4.40 * 10^{-5}$	$4.02 * 10^{-4}$
Earthquake	1.7M (200)	$6.08 * 10^{-1}$	$1.94 * 10^{3}$
Power Demand	10M (4000)	$8.52 * 10^{-3}$	$1.29 * 10^{-1}$
Chicken	9M (1000)	$1.96 * 10^{-3}$	$1.70 * 10^{1}$

precisely by single-precision floats. It may be possible to reduce the error of $SCAMP_{SP}$ for more types of data, but we leave this task for future work.

4.4 Case Studies in Seismology

Figure 3.18 and 4.1 show that motifs are important to many domains. We limit our case studies reported in this chapter to seismic data, which provides information about Earth's interior structure and processes. We define seismic data to be any recorded motion (e.g., displacement, velocity, acceleration) measured using seismic instruments at the Earth's surface. Detected and located seismic events (i.e. earthquakes) can be used for studying earthquake source processes and source physics, fault behavior and interactions, for determining Earth's velocity structure, and to constrain seismic hazard [34]. Many of these applications benefit from detection of smaller events, which can be missed due to insensitive detection algorithms, or human analyst error [16]. Improvements to seismic data instruments, networking and data management, and reductions in cost, have resulted in a power

law increase in seismic data volume [48]. Probing this huge volume of data is an ongoing challenge.

Performing query searches for seismic data can increase the detectability of seismic events by one order of magnitude [76][89]. However, this method requires a priori known queries (often referred to as 'waveform templates' in seismology) as input.

Although waveforms of events in a local earthquake catalog can be used, this relies on suitable events being present in the catalog. While an 'autocorrelation' motif discovery method can identify suitable queries, it is expensive computationally in terms of memory and time [18][84]. The analysis in [18] was restricted to one hour of data, which limited the number of discoverable motifs.

Other studies have performed motif discovery by converting seismic time series to small and dense proxies, and computing a Locality-Sensitive Hash (LSH) [13][110][80], an approximate and reduced-dimension nearest neighbor search. This approach was ∼143x faster than autocorrelation for one week of continuous data, but produced false positive and false negative results [110]. In addition, LSH requires the careful selection of multiple, data set-specific tuning parameters, a process that requires visual inspection and validation against the results of other methods.

In contrast, SCAMP can exactly search datasets that can only be searched approximately using current methods. We consider the milestone of one billion data points (∼579 days, ∼1.5 years) of seismic data with a 20 Hz sample rate. In two examples, we demonstrate how and why transitioning motif discovery timescales from hours of data to years of data is a potential game changer for the field of seismic data mining.

4.4.1 Detecting Foreshocks and Aftershocks

The town of Parkfield, located on the San Andreas fault in central California, experienced four magnitude ∼6 earthquakes in the 20th Century: 1901, 1922, 1934 and 1966 [10]. A repeat event was predicted to occur between 1985 and 1993, spurring the "Parkfield Earthquake Prediction Experiment", which tried to capture the earthquake with the best available instrumentation. The actual event (the "mainshock") occurred 'late' in 2004, and was recorded in extraordinary detail by the low-noise, borehole seismometers of the Parkfield High Resolution Seismic Network (HRSN) [10][54]. Many of these earthquakes were detected and cataloged in real-time at the Northern California Earthquake Data Center (NCEDC) by an automated procedure, and quality checked for false positives by human analysts.

We use this catalog as a reference to investigate i) whether the HRSN data contain information on any aftershocks that were not included in the NCEDC catalog, and ii) whether there was any change in behavior before the mainshock, we ran SCAMP on 580 days (1,002,240,008 points) of data from Parkfield. We use 20 Hz horizontal component seismic data (from 28-11-03 to 9-7-05) from the HRSN station VCAB, centered on the 2004 Parkfield mainshock time (i.e. 28-9-04). We set the query length at 100 samples (5 seconds). We band-pass filtered the data between 2 and 8 Hz, a frequency range that can detect low signal-to-noise ratio earthquakes.

Figure 4.8 shows a zoom-in of two sections of the waveform and their corresponding MPs. The motifs for aftershocks of the Parkfield earthquake have a very characteristic shape. The MP drops abruptly as the query window begins to capture the beginning of the

earthquake waveforms, followed by a gradual increase back to the background noise level, indicating that the two waveforms being compared have similar shapes at their beginnings, and dissimilar shapes at their ends.

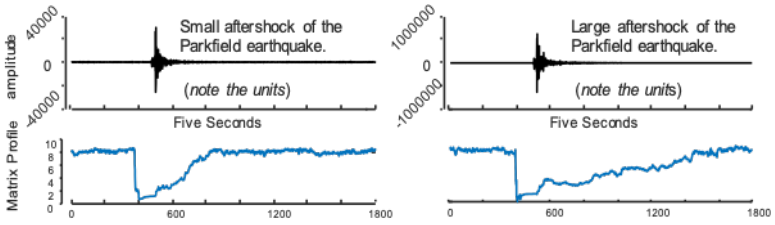

Figure 4.8: Examples of a waveform snippet (top) and corresponding MP shape (bottom) for aftershocks of the Parkfield earthquake. left) a small aftershock. right) a larger aftershock with a waveform amplitude that is three orders of magnitude larger.

The first arrivals (first motions) of seismic waves have polarities (either up or down) that reflect both the mechanism of the earthquakes that generated them and their location relative to the station. The initial drop in the MP indicates the waveforms have the same first motion polarity. The next few seconds of arrivals to the station include reflections, refractions and reverberations of seismic waves – collectively referred to as the seismic 'coda' – which are much more sensitive to differences in earthquake location, and therefore much less similar between pairs of events [3]. The duration of the gradual increase in the MP is longer for the larger event (Figure 4.8.right), consistent with the empirical relationships of signal duration (and coda length) with event magnitude [53][20]. We propose two important applications of MP results to seismology: ii) The abrupt initial drop of the MP can select the first motions of seismic events, which is an ongoing challenge in seismology [69][81]. (ii) The length of the MP valley from the sudden drop to its recovery can help to measure the coda length, which correlates with earthquake magnitude [20][53].

Next, we performed an event-detection experiment using a MP containing the Pearson Correlation Coefficient (MPCC, for short). Pearson correlation is bounded in the range [-1,+1], can be trivially converted to Euclidean Distance, and is widely used in seismology studies [77][67][88]. We count the number of MPCC peaks separated by at least 100 samples (5 seconds) to prevent overcounting the same earthquake when multiple peaks are present for one event. Long traces of seismograph data often contain repeated patterns corresponding to special types of sensor noise; these are easy to filter, as they create near perfect motifs. We count the number of MPCC peaks in the range [0.90, 0.99].

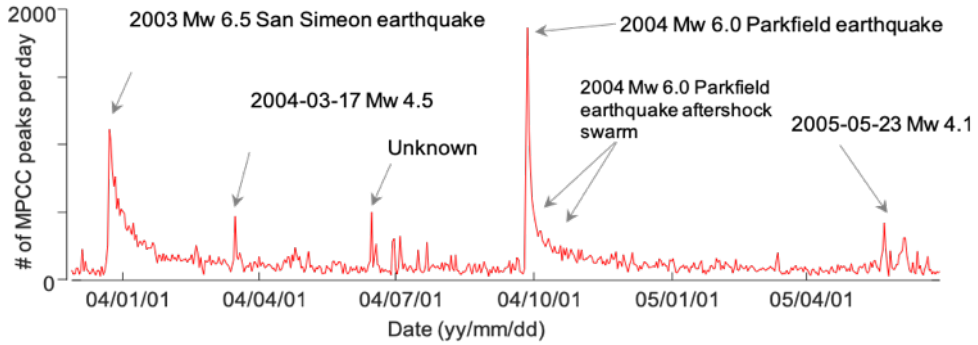

Figure 4.9: Daily number of discovered motifs for 580 days of data centered on the Parkfield earthquake (04/09/28), measured on the horizontal component of station VCAB, located 10 km from the epicenter. Motifs are selected based on the peak MPCC values.

Figure 4.9 shows the number of MPCC motifs per day for our 580 days of VCAB data. Although we targeted the Parkfield earthquake, we detected other nearby earthquakes and their aftershocks, notably the 2003 Mw 6.5 San Simeon event, and two other moderate (Mw 4.0–4.5) earthquakes nearby. A series of motif peaks in the lead-up to the Parkfield mainshock (around 04/07/01) do not correspond to events in the regional earthquake cata-

log, and may represent previously undetected foreshock activity; we have reported them to collaborators in seismology to investigate.

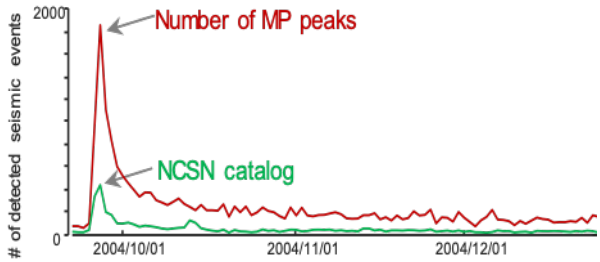

Figure 4.10: The number of events in the USGS NCSN Catalog (green line) and the number of motifs detected using SCAMP (red line) for the Parkfield earthquake aftershock sequence. For the catalog events we considered all events in a box with length 200 km centered on the Parkfield mainshock epicenter. The start of seismicity in this plot is 4 days prior to the Parkfield earthquake

Figure 4.10 compares the total number of motifs in the MPCC range [0.9, 0.99] over the first 90 days of the Parkfield aftershock sequence with the number of catalog aftershocks reported in the NCEDC catalog. This analysis reports ∼16x more detections than those reported by the NCEDC. Some of these thresholding-based detections may be station artifacts, but visual inspection suggests that they account for less than 5% of the events.

We also fit the Omori-Utsu aftershock rate equation [101] to the detected and catalogued aftershocks of the Parkfield earthquake.

Figure 4.11 shows that the number of motifs per day fit the Omori-Utsu law almost perfectly. Values retrieved from the Omori-Utsu rate equation can provide information about the physics of the mainshock [42] and also even can be used for forecasting large aftershocks [72].

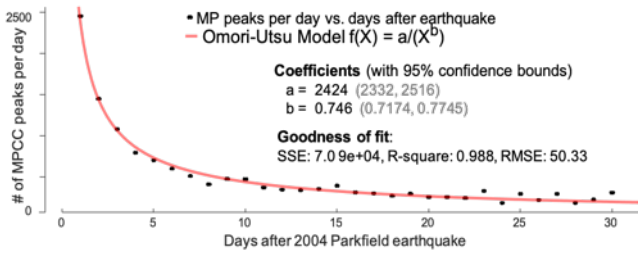

Figure 4.11: A fit of an Omori-Utsu relationship [101] (i.e. the law that describes aftershock rate behavior) to the number of motifs per day for the first 30 days after the Parkfield mainshock. The R-squared of 0.988 indicates a very good fit and shows how the number of motifs can describe the expected aftershock behavior almost perfectly.

4.4.2 Detecting Subtle Seismic Motifs

Low frequency earthquakes (LFEs) are seismic events that occur deep in the crust and typically have very low signal-to-noise ratio signals. LFE recurrence is a proxy for movements at the roots of fault zones, and may be useful in short-term earthquake forecasting [93][71][85]. LFEs have been observed in the Cascadia subduction zone, where the Juan de Fuca plate subducts beneath the North American plate, from coastal Northern California to Vancouver Island. This 'megathrust' fault has the potential to produce great (magnitude $\sim$9) earthquakes [8], motivating LFE detection in this region. Their low signal-to-noise ratios make detecting them challenging and time consuming (e.g., requiring sophisticated methods and visual inspection; [15][92][18]).

In order to see if we can detect these novel events in this region, we ran SCAMP on 579 days of data (start date 2006/03/01) for the vertical component of station I02A, located near Mapleton, OR. We band-pass filter these data at 2–8 Hz and resample them to

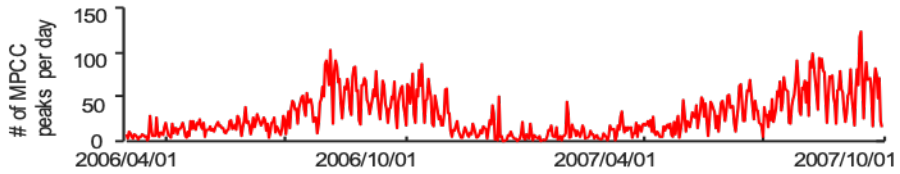

Figure 4.12: Discovered motifs for 579 days of seismic data recorded on the vertical channel of station I02A, located near Mapleton, OR. The number of discovered motifs based on MPCC thresholding method shows two six-month periods were detected motifs gradually increase, that start in mid-2006 and mid-2007. We believe many of these motifs are low frequency earthquakes (see Figure 4.13).

20 Hz. We set the query length to 200 (10 seconds), based on the length of LFE templates used in previou[15].

Figure 4.12 shows the motif density over time for this experiment. The number of motifs starts to increase around August 2006 and decrease in November 2006, and again increase in June 2007 and start to decrease around October 2007. We visually inspected some of these motifs (in both time and frequency domain) and classified them in four categories: i) regular earthquakes (less frequent, Figure 4.13. left.) ii) weather or human related signals (frequent), iii) Station artifact (less frequent), iv) LFE-like signals (frequent, Figure 4.13.right). Confirming a signal to be LFE is not easy, typically requiring detection at several stations and visual inspection of its frequency spectrum. In Figure 4.13 we show a discovered motif that was confirmed as a true LFE in [15]. Note that the MP for the LFE is not as low as regular earthquake but much lower than the background noise (Figure 4.13).

In general, we detect fewer than 150 motifs per day in this dataset. This means that in order to discover LFEs a seismologist needs to inspect fewer than 150 sub-windows per

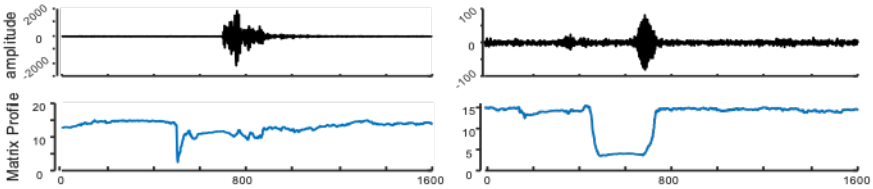

Figure 4.13: left) An example of an earthquake waveform snippet (top) and MP shape (bottom) in the vicinity of a discovered motif for a 'regular' earthquake. right) A waveform snippet and corresponding MP from a confirmed LFE (identified by [15]).

day of data, a task that would take minutes to perform. In contrast, the traditional visual inspection method for detecting LFEs (e.g., brute force checking [92]) requires inspection of thousands of sub-windows (e.g., 17280 sub windows with a 5 second skip), potentially taking hours for each day of seismic data. Running SCAMP before searching for these subtle and important motifs could potentially provide a large time savings for seismologists and make their discovery much easier in this domain.

These results were obtained by post-processing an MP produced by SCAMP; possibilities for further refinement remain open. These results show that SCAMP can detect LFEs, and has the potential to more generally explore the seismicity of the southern Cascadia subduction zone and other similar regions. We believe that SCAMP has a rich future in seismic data mining – a discipline that traditionally suffers from false negatives – and other domains that produce time series.

4.5 Chapter 4 Summary

SCAMP exactly searches for motifs in time series at the data-center scale. To the best of our knowledge, this work is the first time any research effort has reported performing a

quintillion exact pairwise comparisons on a single time series dataset. Likewise, we believe this to be the first work to do exact motif search on more than one year (1.59 years to be precise) of continuous earthquake data. All code has been made freely available to the general public [114], whom we invite to confirm, extend, and exploit our efforts.

Chapter 5

LAMP - An Approximate Matrix Profile for Massive Archives and Fast Moving Streams

5.1 Introduction

By efficiently computing all of the "essential" distance information between subsequences in a time series, the Matrix Profile makes many analytic problems, including classification and anomaly detection, easy or even trivial. However, for many tasks, in addition to archives of data, we may face never-ending streams of newly arriving data. While there is an algorithm to maintain a Matrix Profile in the face of newly arriving data, it is limited to streams arriving on the order of one Hz and with small archives of historical data. However, in

domains as diverse as seismology, neuroscience and entomology, we may encounter datasets that stream at rates that are orders of magnitude faster.

In this chapter we introduce LAMP, a model that predicts, in constant time, the Matrix Profile value that would have been assigned to an incoming subsequence. This allows us to exploit the utility of the Matrix Profile in settings that would otherwise be untenable. While learning LAMP models is computationally expensive, this stage is done offline with an arbitrary computational paradigm. The models can then be deployed on resource-constrained devices including wearable sensors. We demonstrate the utility of LAMP with experiments on diverse and challenging datasets with billions of datapoints on a simple desktop machine. Using LAMP, we can achieve more than 10000x speedup over exact methods on the same data.

In previous chapters we saw ways to compute the Matrix Profile using tiling and batching. However, the MP can also be computed incrementally, enabling streaming versions of algorithms which exploit the MP.

STOMPI [109] is the current state of the art algorithm for maintaining the matrix profile on streaming data. However, STOMPI has a problem: the time required to update the MP slowly grows as a function of how much data we have seen. Suppose we start monitoring a new 5 Hz process at midnight on Sunday. Initially, we can use STOMPI to maintain the MP, and have plenty of cycles to spare. However, by Wednesday at 10:25 AM, when we have seen just over one million datapoints and we can no longer maintain the MP fast enough , the next datapoint will arrive before STOMPI is finished updating the matrix profile for the last datapoint.

We can push back this time horizon with faster machines, but the reprieve is temporary. At some point, the growing computational demands will outstrip our resources. To make this concrete let us preview two real-world applications of our system that we will later revisit in our experiments. In Figure 5.1.top we show a classification problem for telemetry for insects. The recording apparatus produces a snippet that we must classify in to one of several classes. We have just 1/100th of a second to do this, before the next snippet arrives.

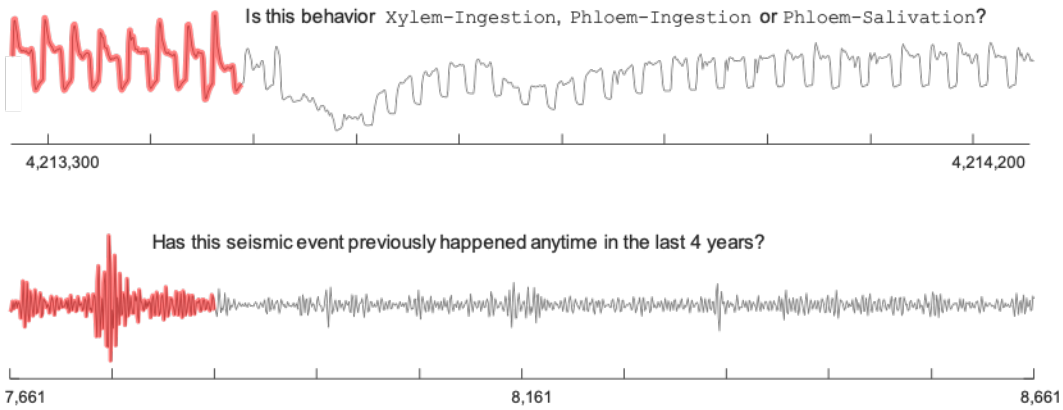

Figure 5.1: Two time series subsequences (shown in red) that need to be quickly processed. top) An example of data from an insect EPG (Electrical Penetration Graph) apparatus. bottom) An example of a trace from a seismograph.

In Figure 5.1.bottom we show a snippet from a seismograph. Here the sampling rate (after some inline processing) is slower, with new snippets arriving every 1/20th of a second. However, to answer the question posed, we need to compare this data with four years of data, or $2.53 * 10^9$ datapoints.

The problem is exacerbated by the fact that we would like to deploy MP-based algorithms on embedded devices with very little computational power. This would potentially

allow analytics to be done "at the edge" [73], reducing the network and power overhead of transmitting data.

In this work, we propose to solve this problem by introducing a Learned Approximate Matrix Profile (LAMP), which enables constant time approximation of the MP value given a newly arriving time series subsequence. With this approximate value, we can do most of the analytics based on the MP, including anomaly detection and classification.

5.2 Related Work and Background

In this section, we first introduce all necessary definitions before considering related work.

5.2.1 Definitions

Chapters 3 4, and [109] defined the matrix profile in terms of the Euclidean distance between z-normalized subsequences. However, in this chapter, we define the matrix profile in terms of the Pearson correlation. This is because it creates results limited to the intuitive range of [-1, 1]. For example, seismologists may prefer to filter out weakly matching sequences for some analytic task, perhaps by setting a correlation threshold to say 0.8 [90]. Working with correlation allows them to reuse such a threshold on multiple datasets, without having to worry about the sampling rate of the length of the subsequences. In contrast, for Euclidean distance, any threshold discovered would have to be recalibrated for new sampling rates or subsequence lengths.

Given a query subsequence $T_{i,m}$ and a time series T, we can compute the correlation between $T_{i,m}$ and all the subsequences in T. We call this a correlation profile:

Definition 6 *A correlation profile C_i corresponding to query $T_{i,m}$ and time series T is a vector of the Pearson correlations between a given query subsequence $T_{i,m}$ and each subsequence in time series T. Formally, $C_i = [c_{i,1}, c_{i,2}, ..., c_{i,n-m+1}]$, where $c_{i,j} (1 \leq j \leq n-m+1)$ is the Pearson correlation between $T_{i,m}$ and $T_{j,m}$.*

It is important to recognize that using correlation does not change the information contained from the previous matrix profile (Definition 4), as the Pearson correlation can be converted to z-normalized Euclidean distance in constant time using Equation 4.5. Moreover, the ranking of all the top-K nearest neighbors to a time series is *identical* under Pearson correlation and between z-normalized Euclidean distance.

Once we obtain C_i, we can extract the nearest neighbor of $T_{i,m}$ in T. Note that if the query $T_{i,m}$ is a subsequence of T, the *ith* location of correlation profile C_i is 1 (i.e., $c_{i,i} = 1$) and close to 1 just to the left and right of i. This is the same trivial match discussed in Section 3.1.4. Like before, we avoid such matches by ignoring an "exclusion" zone of length $\frac{m}{4}$ before and after i, the location of the query. In practice, we simply set $c_{i,j}$ $(i - \frac{m}{4} \leq j \leq i + \frac{m}{4})$ to negative infinity, and the nearest neighbor of $T_{i,m}$ can thus be found by evaluating $max(C_i)$.

We wish to find the nearest neighbor of every subsequence in T. This nearest neighbor information is stored in two "meta time series", the matrix profile and the matrix profile index:

Definition 7 *A matrix profile P of time series T is a vector of the Pearson correlation between every subsequence of T and its nearest neighbor in T. Formally:*

$P = [max(C_1), max(C_2), ..., max(C_{n-m+1})]$, *where* C_i $(1 \leq i \leq n-m+1)$ *is the correlation profile* C_i *corresponding to query* $T_{i,m}$ *and time series* T.

The ith element in the matrix profile P tells us the Pearson correlation from subsequence Ti,m to its nearest neighbor in time series T. However, it does not tell us the location of that nearest neighbor; this is stored in the companion matrix profile index, which in this case is computed very similarly to the previous Definition 5, except we use max instead of min.

Definition 8 *A matrix profile index* I *of time series* T *is a vector of integers:* $I = [I_1, I_2, ..., I_{n-m+1}]$, *where* $I_i = jif c_{i,j} = max(C_i)$.

Figure 5.2 shows the relationship between correlation matrix, correlation profile (Definition 6) and matrix profile (Definition 7). Each element of the correlation matrix $c_{i,j}$ is the correlation between $T_{i,m}$ and $T_{j,m}$ $(1 \leq i, j \leq n-m+1)$ of time series T.

Figure 5.3 shows a visual example of a correlation profile and a matrix profile created from the same time series T.

Note that as presented above, the matrix profile can be considered a self-join [109]: for every subsequence in a time series T, it records information about its (non-trivial-match) nearest neighbor in the same time series T. However, as mentioned before in Section 3.1.4 we can trivially generalize it to be an AB-join [109]: for every subsequence in a time series A, it records information about its nearest neighbor in time series B. Note that A and B can be of different lengths, and that in general, AB-join $\neq$ BA-join. This becomes important to the following definitions.

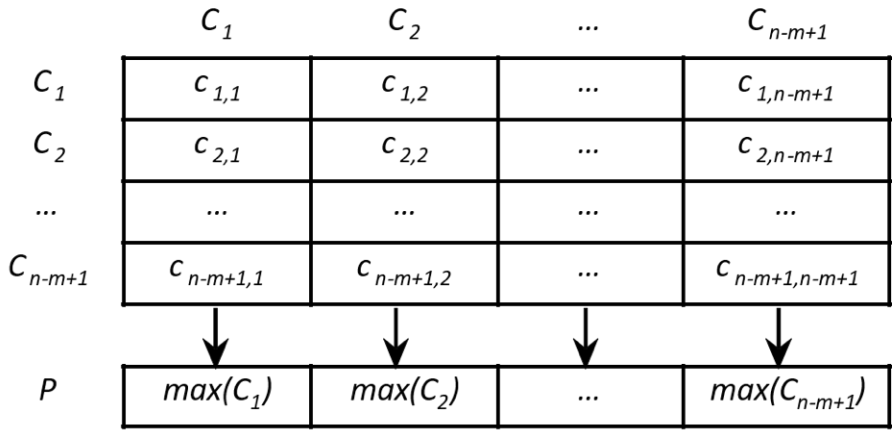

Figure 5.2: The relationship between the correlation matrix, correlation profile and matrix profile. A correlation profile is a column (also a row) of the correlation matrix. The matrix profile stores the maximum (off diagonal) value of each column of the correlation matrix; the location of the maximum value within each column is stored in the companion matrix profile index.

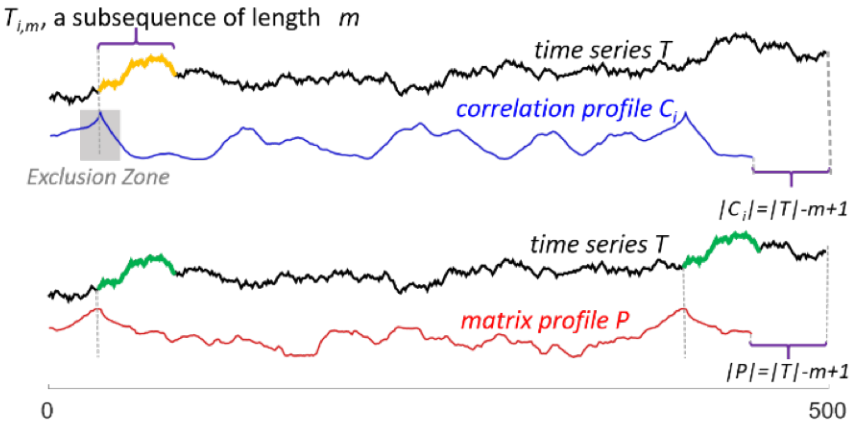

Figure 5.3: top) A correlation profile C_i created from $T_{i,m}$ shows the correlation between $T_{i,m}$ and all the subsequences in T. The values in the dark zone are ignored to avoid trivial matches. bottom) The matrix profile P is the element-wise maximum of all the correlation profiles (C_i is one of them). Note that the two highest values in P are at the location of the 1st motif in T . (Adapted from Figure 3.4).

We are now able to introduce the definitions immediately relevant to the problem at hand. First, we consider the output of the STOMPI algorithm [109], which is the exact

matrix profile for all data seen up to the current time in a streaming setting. This will serve as our ground truth or oracle.

Definition 9 *An Oracle Matrix Profile (OMP) of a stream S is the matrix profile of the entire stream; it encodes the nearest neighbor for all subsequences in the history of the stream, where the nearest neighbor can be any observed subsequence of S.*

Given a new set of k consecutive subsequences observed from S: the OMP can be updated via STOMPI in time $O((S + k)k)$, which is the time it takes to compute the AB-join between S and k and the self-join of k. For data with a very low sample rate, it might be enough to simply maintain the OMP. However, in many cases this is untenable because the cost of maintaining the OMP grows as more data is observed. Therefore, we assume we have a representative subset of S that we can use as a proxy.

Definition 10 *A Representative Matrix Profile (RMP) of a stream S is the matrix profile of the entire stream, where the nearest neighbor can only occur in some representative subset R consisting of observed subsequences of S. Formally, RMP is the AB-join between S and R where RMP encodes the nearest neighbor of each subsequence of S in R. Note that an exclusion zone must be applied to each subsequence in R when comparing to its 'original copy' in S.*

We can update the RMP in time $O(Rk)$ which is the time it takes to compute the AB-join between R and k. Note that the time complexity per update no longer depends on the entire history of the stream. For some applications this might be enough. However, for applications with a high sample rate, large R, or on systems with low computational power, this will still likely be above our compute budget and we will need something better.

Definition 11 *A Learned Approximate Matrix Profile (LAMP) of a stream S is the output of a learned model that compresses R into a fixed-size compressed representation. It approximates the RMP of S.*

Using this definition, we can now perform updates in $O(k)$, no longer depending on the size of the representative dataset.

5.2.2 Motivation and Formal Problem Statement

Assume we have a continuously arriving stream of time series from a sensor. We may wish to take the most recent subsequence of length m and compare it with an archive of previously collected data. There are multiple reasons why we may wish to do this, including:

- **Classification**: We may have partitioned our archive of previously collected data into labeled subsets, for example wild-type — mutant [17] or ingestion — probing — salivation [108]. In this case we have an implicit nearest neighbor classifier.
- **Anomaly Detection**: In some domains, we can expect that all newly arriving subsequences should be close to a pattern we have already observed. A pattern that is not (formally a "time series discord" [6]) may signal the discovery of an anomaly.
- **Segmentation**: In [37] it was shown that a very competitive time series sematic segmentation algorithm can be built on top of the Matrix Profile.

A more formal problem statement is:

Problem Statement: Given a streaming time series and representative subset of data from that stream, subject to the constraint that data must be analyzed at the time

of arrival, approximate the matrix profile values associated with this newly arriving data, such that they closely approximate the matrix profile values that would be produced by existing exact methods.

5.2.3 Dismissing Apparent Solutions

Before introducing LAMP, here we will take the time to dismiss some apparent solutions to the task at hand.

- **Indexing**: Would it be possible to just index the data, and perform a nearest neighbor search for each arriving subsequence? Recall that the seismology example shown in Figure 5.1.bottom would require us to index $2.53 * 10^9$ datapoints. The fastest query times for datasets approaching this size are three to four orders of magnitude slower than our required processing rate [32]. Moreover, virtually every indexing techniques takes a variable and unpredictable amount of time to answer queries. Thus, even if we had a much slower arrival rate where the index could keep up on average, and we had a highly optimized index running in main memory, it is always possible that we could see multiple slow-to-process subsequences in a row, and therefore run out of time.

- **Dictionary Building/Numerosity Reduction**: Could we not just build a compact "dictionary" of events and brute force search it in the time allowed? There are many papers that suggest something like this, and it is good idea in limited circumstances. For example, it seems to be possible to explicitly build a full dictionary of heartbeats; several papers have explicitly suggested this "We model heartbeats by dictionaries.." [19]. However, heartbeats are a relatively easy case, as there are algorithms that

can robustly extract individual phase-aligned beats. In contrast, we are interested in datasets where this is not possible in general, because the target behavior is highly polymorphic, and only weakly labeled. Consider Figure 5.4, which shows some examples of a single behavior from an insect. We know it reflects a single behavior because an entomologist labeled the entire five-minute session with the label Xylem-Ingestion. However, it not clear that we could build a dictionary to summarize this class, either with an algorithm or using significant human labor.

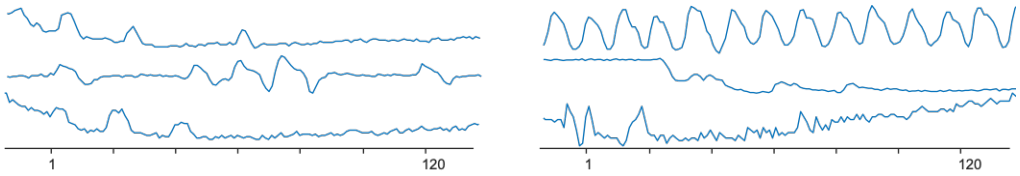

Figure 5.4: Six random examples of insect Xylem-Ingestion behavior, from a single insect, taken from a five-minute window.

In a sense, LAMP is implicitly both indexing and dictionary building. The intuition behind LAMP can be summarized as: If a data object is conserved in the training data (dictionary building) then make sure it is represented in the LAMP model (index). However, unlike indexing, LAMP can return an answer in strictly bounded constant time, and unlike dictionary building, LAMP does not need carefully curated data.

5.2.4 Related Work

LAMP touches on many aspects of data mining, time series analysis, classification, streaming data and deep learning. However, we believe that there is no direct competitor to LAMP currently. While there exists a technique to incrementally update the Matrix Profile [109],

it is limited to settings where the update rate is relatively slow, on the order of ∼1 Hz. As the original authors point out [109], there are many domains where this is more than sufficient. However, domains in medicine, seismology and life sciences (i.e. entomology) can produce data at least two orders of magnitude faster than this. While one instance of LAMP uses a deep neural network, it is important for us to note that we are not claiming any contribution to deep learning. We simply assume that the current state-of-the-art can be plugged into our framework.

5.3 Method

In order to avoid ever-growing computational cost as more data is observed, we will assume that we have some representative time series, R, observed from the stream. For example, the insect data we empirically consider in 5.4 is collected each day, seven days a week in ten to sixteen-hour sessions. We can take a single day of this data, and use it as our R, for all sessions recorded on subsequent days.

Given R, we can generate the RMP (Definition 10) for the stream. The RMP is illustrated in Figure 5.5. If our comparison data is truly representative of the stream, then this RMP will very closely resemble the oracle OMP (Definition 9).

If R is small and our sample rate is low enough, say, under one million datapoints, with a sample rate of 1 Hz, then we are done. We can approximate the matrix profile values of new subsequences in time $O(||R||)$. However, if the representative dataset is large, or we are working with low power hardware, then the computational complexity would likely still be above our compute budget for typical streaming rates.

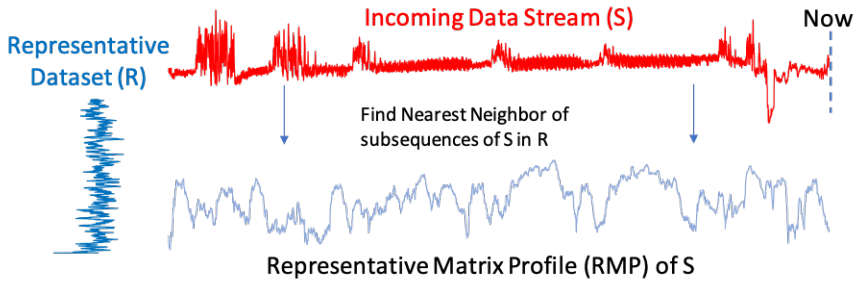

Figure 5.5: Illustration of comparisons to compute the ab-join MP with a representative dataset (Linear in the length of R)

Thus, in order to truly have a useful method in the general case, we need an algorithm that does not depend on the full size of the representative dataset. To this end, we implement LAMP (Definition 11), which models the representative dataset in a compressed, fixed-memory-size model, which requires a fixed time budget to process each arriving datapoint. Figure 5.6 illustrates this idea.

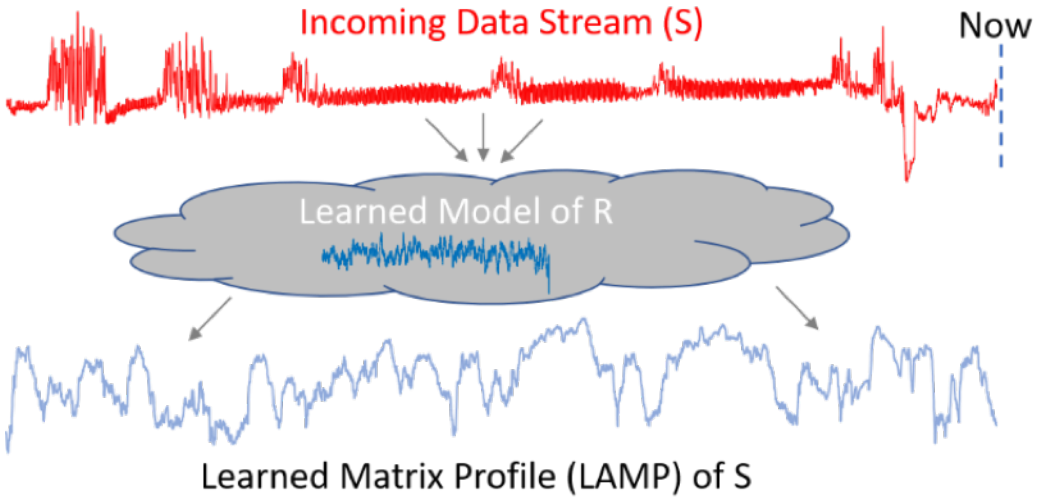

Figure 5.6: Illustration of approximation of the matrix profile using a model learned from a representative dataset.

There are many learned models we can choose to summarize the training data; we choose to highlight two of these in this work.

5.3.1 Top K Diverse Motifs

Table 5.1 explains our method for extracting the Top-k diverse motifs from a training dataset. In line 1 we compute its exact MP using SCAMP (see Chapter 4), then in line 2-3 we sort the MP to generate a list of the top motifs. Then in lines 5-11 we create a model which contains a set of diverse motifs such that no pair is closer to each other than the diversity threshold.

There are other methods for selecting a diverse set of k motifs. For example, prior work investigated the k-diversification problem for time series [33]. We leave such considerations for future work.

There are several advantages to using the Diverse Motifs model as the subroutine for LAMP. It is highly interpretable; every subsequence in the model (everything that the model "knows") can be directly visualized. Additionally, adding examples to the model is as simple as appending to a list.

Moreover, because the motifs are sorted by their utility, we can use the Diverse Motifs model as an anyspace model. An anyspace model is the analogue of an anytime algorithm [109], but with memory as the limiting factor. For example, suppose we create a 10,000 motif model to use in an insect monitoring task in a lab (See Figure 5.12). However, later the entomologist has the idea to run experiments in the field with a small memory limited device such as a raspberry pi, which only has space or compute resources for say

Table 5.1: Training and Predicting using the Top-k Motifs LAMP Representation

Procedure Train

Require: time series t, subsequence length m, number of motifs to extract k, diversity threshold d.

```
MP := SCAMP(t, m)
indexes := sequence(0,length(MP))
indexes := argsort(indexes, MP)
model := [], count = 0
while count < length(MP) and length(model) < k do
    considered := t[indexes[count]:indexes[count] + m]
    if IsDiverse(considered, model, d) then
        model.append(considered)
    end if
    count := count + 1
end while
return model
```

Procedure IsDiverse

Require: c (candidate sequence to check), curr (list of current motifs), t, (diversity threshold)

```
for sequence in curr do
    if correlation(c, sequence) ¿ t then
        return false
    end if
end for
return true
```

Procedure Predict

Require: s, (observed sequence), model (top-k motifs returned by **Train**)

```
maxcorr := -1;
for motif in model do
    corr := correlation(motif, sequence)
    if corr > maxcorr then
        maxcorr := corr
    end if
end for
return maxcorr
```

2,500 motifs. We can simply truncate off the bottom $\frac{3}{4}$ of the motifs and push the model onto the smaller device. The main disadvantage of this method is that it is essentially uncompressed. Since the size of the model affects the number of operations required to

perform the prediction, performing inference when k is large can be slow and prohibitively expensive to run in real-time.

5.3.2 Neural Networks

Using a neural network as the basis of the LAMP model has many advantages. We can utilize any of the infrastructures built up around deep learning over the last several years, including GPU optimized code, embedded platform support, and ongoing research in accuracy/speed tradeoffs, which can allow us to adapt to a stream's sample rate according to our platform.

While LAMP is agnostic to the actual network used, in this work we use the simplified version of Resnet [46] proposed by [106] for time series classification, but with the activation on the output layer changed to sigmoid to enable regression. We also modify the input and output of the Resnet model to support multiple predictions at once. i.e. Each of our inputs consists of J z-normalized subsequences of length M from the data, extracted with stride S. This procedure defines an extraction window in the time series, W, where $||W|| = JS + M - 1$. We can slide W across the time series and extract a new input for the neural network for each position of W. Following this logic, each input to the neural network is a vector of length M with J channels, where we set M as the subsequence length parameter of the matrix profile. For each input, the neural network outputs JS LAMP values, one for each subsequence in W. Figure 5.7 shows the outline of our scheme.

This input scheme has three main advantages over a single subsequence input scheme:

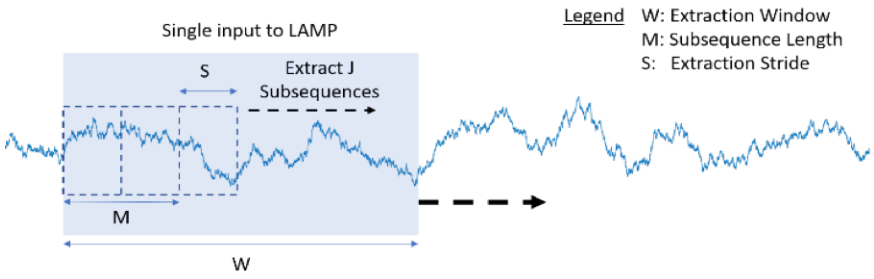

Figure 5.7: LAMP neural network input scheme

- Reduces overfitting by increasing the dimensionality of the output space. Intuitively, a larger output dimension provides regularization and leads to smoother predictions.
- Enables faster processing by GPUs and CPUs by exposing additional parallelism through the added dimensionality.
- Enables the convolutional network to learn short-term time dependencies in the data.

It is important to note that when the subsequence length is very long, the inputs to the neural network also get large. Though it is possible to perform subsampling and other types of dimensionality reduction on the input before sending it to a LAMP model, we have found that the most effective way to reduce the amount of input to the model is to increase the extraction stride S. In almost all applications, this is a reasonable assumption.

For example, if a classifier correctly predicts that you are running at time 17sec, and that you are running at time 19sec, it is a reasonable assumption that you were also running at all times in-between. For very long subsequence lengths, there is a large overlap between the information contained in consecutive or close-by subsequences, so a moderate increase in the extraction stride typically causes an imperceptible accuracy loss in these cases.

For the experiments in this paper, we set $W = 256$, $S = 8$, $J = 32$. We set the learning rate to $1e - 3$ and we use the Adam [51] optimizer for stochastic gradient descent with a batch size of 32. We optimize the network for the mean squared error loss between the predicted and exact RMP values for our training data. These all reflect common values/practices in the literature. We did not carefully optimize the model, as we wish to demonstrate the robustness of our overall system. The network is implemented in Keras and available at [113].

5.3.3 Configuring the Model Size

It is useful to consider how to select the size of a LAMP model, as this can be done deterministically before model construction. Given the computational capability $\mathbf{c}$ (FLOP/s) of a system and the sample rate $\mathbf{r}$ (Hz) of a stream we can compute the maximum size of a model, in terms of the number of FLOPs possible per inference step using the equation $FLOPs = \mathbf{cr}$. Once we know our limitations, we can choose a model appropriate for our specific use case.

It is also important to note that in many of these applications we do not need the result *immediately* even in a real-time application. For example, if our sample rate is 100Hz, perhaps we don't need to make decisions based on every single new datapoint, but only once per second. In this case, we can process multiple subsequences at once via batching, like in our neural network input scheme, which is more efficient computationally, and can help give additional context to our predictions.

5.4 LAMP Evaluation

To ensure that our experiments are reproducible, we have built a website [113] which contains all data/code/raw spreadsheets for the results, in addition to many experiments that are omitted here for brevity. Unless otherwise stated, all experiments were run on a system with an Intel Core i7-8700K CPU and 32GB RAM.

For neural network LAMP, we used the parameters discussed in Section 5.3.2 for all experiments. Clearly tuning the neural networks could produce improved results, however we wanted to demonstrate the generality of LAMP models and to show that they can work well "out of the box". Similarly, for Diverse Motifs LAMP we use a hard-coded diversity threshold of 0.95 unless otherwise noted.

In the following section we evaluate LAMP in the most direct way possible. Recall that the goal of LAMP is to predict the value that the much slower full Matrix Profile algorithm would have produced, thus we can both visualize and measure the difference between the OMP and LAMPs output. However, in some sense this is an indirect measurement for most practitioners. They typically only care about the classification or detection accuracy of their higher-level tasks which would exploit LAMP. Thus, in the remainder of the paper we will offer detailed case studies to demonstrate that LAMP can offer real-time performance even in challenging scenarios.

5.4.1 LAMP method evaluation

In Table 5.2, we compare the performance of various model types with a subsequence length of 100 on various architectures. The values in the table are measured in subsequences per second. The first two rows show the Diverse Motifs model with various settings for K. We did not implement these methods on the GPU, which is why no results are reported for the Tesla P100. For these models, our implementation was unbatched; it used only a single thread and processed just a single subsequence at a time.

Table 5.2: LAMP Inference Performance for m = 100

Diverse Motifs Inference Rates (Hz)			
Params	Tesla P100 GPU	CPU (i7-8700K)	Raspberry Pi 3
K = 1000	N/A	4852	434.8
K = 60000	N/A	403	16.9
Neural Network Inference Rates (Hz)			
Params	Tesla P100 GPU	CPU (i7-8700K)	Raspberry Pi 3
J = 1, S = 1, Batch = 1	125	200	9.2
J = 32, S = 8, Batch = 1	51.2K	85.3K	2782
J = 32, S = 8, Batch = 128	482K	206K	5461

The bottom 3 rows show the results for our neural network scheme with various levels of batching. The first row is completely unbatched. The neural network is shown every subsequence individually and predicts a matrix profile value for each one. The second row uses the default settings that we presented in the Section 5.3.2 with an input of 32 subsequences with stride 8, each inference produces 256 LAMP values. This enables increased efficiency and other advantages described in Section 5.3.2. The last row uses a second level of batching where the neural network inputs are batched. Depending on how quickly decisions must be made, a user can choose a method of batching to suit their

constraints. Differences in speed between single input and batched input are only because of the added data locality and dimensionality, which allow for exploiting multiprocessor and SIMD architectures. As mentioned previously, LAMP model inference time is dataset agnostic, depending only on the model size and input size.

We note that the neural network is much more efficient in general, due in part to a multitude of optimizations implemented by the Tensorflow developers and the deep learning community at large. Given the resources, a more efficient solution to inference with diverse motifs could be developed. However, it is also true that we have not actively optimized the neural network for any particular inference task. Depending on the dataset and other parameters of the problem, it might be possible to also make the neural network significantly faster via speed/accuracy tradeoffs. For example, adjusting the extraction stride S, applying quantization [43] or resource-constrained structure learning frameworks such as Morphnet [39]. We defer such an investigation to future work.

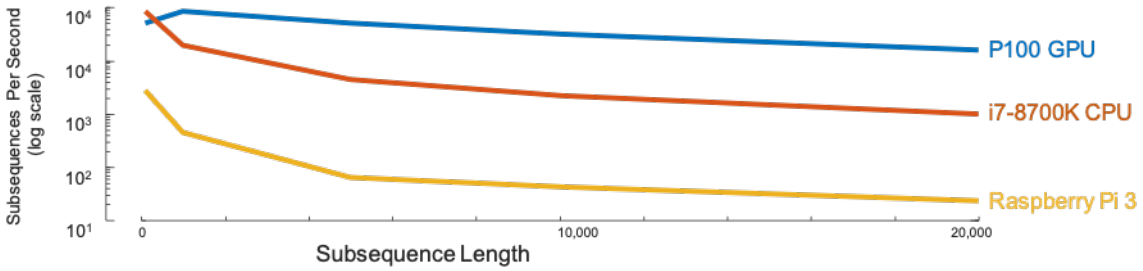

Figure 5.8: Tradeoff between input subsequence length and inference rate for our neural network method on three different architectures.

It is also important to note that because we are extracting subsequences and sending them to the model for inference, the subsequence length parameter influences both the size of the models and inference time, as more FLOPS are required to perform a single

inference using larger subsequences. Figure 5.8 illustrates this tradeoff for our default neural network method defined in Section 5.3.2.

Table 5.3: Comparison of LAMP Model Performance to Oracle

Dataset		Correlation to Oracle (OMP)			
Name	Train/Test Split	Exact RMP	Neural Net	Diverse Motifs	K
Earthquake	20M/10M	.965	.887	.731	60K
Street Mall	59K/17K	.986	.690	.615	130
Insect EPG	2.5M/5M	.973	.959	.625	12K

Table 5.3 illustrates how well our model fits the oracle for various datasets. The RMP's performance can be viewed as a performance measure of the training data. A perfect LAMP model would achieve performance similar to the RMP. Note that as mentioned previously, this is an indirect measurement, as practitioners will be mostly concerned with classification or detection accuracy, which we discuss in the following sections. As mentioned before, there is room for improvement here via parameter tuning, but we have explicitly kept a single set of parameters for Neural Network LAMP to show its robustness. The performance variation for the street mall dataset might be addressed via parameter tuning or the addition of more training data. For the Diverse Motifs model, we report the number of motifs used for that particular dataset. Due to the sensitivity of this parameter and the differences in the size of each of the datasets we evaluated, we could not achieve acceptable performance with the same K across the board.

Figure 5.9 shows a visual comparison of the performance of various LAMP models on a snippet of the seismogram dataset from Table 5.3. Note the smoothness of the neural network model and the improvement of the matches as K is increased for the diverse motifs

model. For this figure, the diverse motif models were generated using a diversity threshold of 0.85, the neural network settings were set to the default.

We have not reported training times for our experiments; however, most of the Neural Network LAMP models were quickly trainable in under an hour or two on a Tesla P100 GPU. The only exception to this is the large dataset presented in the next section, which took approximately 1 day to train on the GPU.

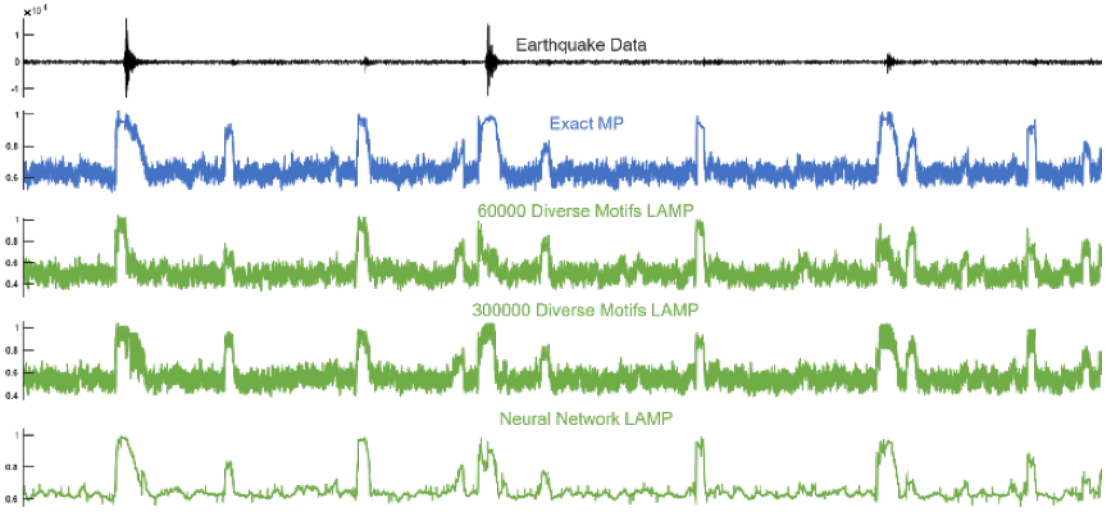

Figure 5.9: Visual Comparison of LAMP methods for a snippet of earthquake data

5.4.2 Case Study in Seismology

Real-time seismic event detection is a primary task in seismology that has a direct impact on earthquake physics, fault behavior and seismic hazard assessment studies [4][87][64]. Most modern seismic networks have implemented real-time streaming of data from their remotely installed instruments to their seismic observatories. There, real-time event detection methods are used to monitor earthquake activity and provide basic information on

event occurrence, timing, and magnitude. This can have a direct impact on seismic hazard assessment and response and early warning systems [4][64].

One of the most common real-time event detection methods is the short-term-average/long-term-average (STA/LTA) method, which computes the ratio between the average seismogram amplitude over a short time window and a longer time window. We expect that an earthquake will cause a sudden increase in the amplitude of the seismic waveform in the short term, leading to a spike in the ratio [91]. This method is widely used due to the ease of implementation and is effective at detecting large and/or close-by events. However, it can fail to detect smaller or more distant earthquakes, whose average waveform amplitudes are close to the noise floor. As a result, we speculate that many small events are missing from seismic event catalogs. Finding these small events is of particular significance in the seismology domain as they have a direct impact on studies of seismic triggering, short-term earthquake forecasting, foreshock and aftershock behaviors, etc. [16].

In some cases, seismologists apply more sensitive detection methods to 'mine' these smaller events from the continuous waveform data. One recent example [83], used query search ('matched filtering' in the terminology used by seismologists) to identify an order of magnitude more small events than had been detected using traditional methods in southern California. Such efforts show the power of a more sophisticated approach, although this improvement in sensitivity is not without cost – in the southern California case, the necessary computation required many hundreds of thousands of GPU hours [83]. Another limitation is that such methods use queries ('template waveforms' in seismology) from the existing seismic catalog in order to search for more events, leaving the possibility

of a remaining population of undetected earthquakes for which there are no appropriate queries in the catalog.

We argue that LAMP is a potential solution for sensitive, rapid and inexpensive real-time seismic event detection. A common sample rate for local earthquake detection is 100 Hz, which is in the range of sample rates for which LAMP can produce the MP for the stream of seismic data in real-time, even on a relatively inexpensive device.

Note that there are many machine learning-based methods that have been proposed for earthquake event detection in recent years [82][111]. These methods are usually trained using existing catalogs, and are based on classified earthquake-noise training data sets. The training data set in this case might pass on the insensitivity of the catalogs to the models. LAMP is trained using the exact MP calculated from one year of data. The MP for one year of data is very sensitive to earthquake occurrence and can increase the number of event detections by $\sim$16 times (see Chapter 4). Rather than the binary classification of earthquake and non-earthquake (noise), LAMP weights waveforms based on a range of MP values (e.g. $\sim$0.5 to 1), based on their similarity to other events.

Here we test LAMP for a real seismic waveform data set and compare the results with the existing catalog of earthquakes. We use a data set from a sensitive, low-noise, borehole seismic station (station name 'VCAB', network ID 'BP') near Parkfield in central California, close to a segment of the San Andreas fault where earthquakes occur frequently.

We train a neural network LAMP model using a 20 percent contiguous sample of this exact MP that we generated for the 1 billion datapoints example in Chapter 4, for 580 days (2003-11-28 to 2005-07-09) of 20Hz seismic data. We then use LAMP to estimate the

Table 5.4: Comparison of Detection Rates of Various Types of Seismic Events For Various Detection Thresholds

	Threshold	W0	W1	W2	W3	W4	Total	"False" Pos.
conservative	0.95	76%	37%	34%	27%	15%	61%	0.56%
::	0.90	89%	65%	60%	54%	32%	79%	1.8%
::	0.85	95%	80%	76%	70%	42%	89%	3.9%
liberal	0.80	98%	90%	88%	83%	47%	94%	7.8%

MP for 5.5 years of data (from 2005-07-10 to 2011-01-01) for the same seismic station. The total inference time for this dataset of around 4 billion datapoints was approximately 20 minutes using the large batch inference configuration from Table 5.2 on a single GPU. By extrapolating the performance of SCAMP in Chapter 4 to 4 billion datapoints, this is a speedup of over four orders of magnitude.

We then use four different thresholds of 0.8, 0.85, 0.9 and 0.95 for detecting motifs. The smaller values are more liberal (sensitive), and more likely to include some false positives. Then in Table 5.4 we compare our detection with earthquake information that we obtained from the Northern California Seismic Network (NCSN) catalog [21]. Here we use two different catalogs to validate the LAMP outputs. The first catalog contains events whose seismic signals have been observed and picked at the station of interest, either by human analysts or by event detection algorithms (e.g., the STA/LTA method). These seismic signal observations are reported with five different weights based on confidence (W0 to W4, from 'very strong detection' to 'weak detection'). In this work, we refer to this catalog as the 'event-station catalog'. The second catalog contains all detected earthquakes, whether or not they were observed at this station, and we refer to it as the 'event-only catalog'. In this 5.5-year period, there were 9546 events in the event-station catalog and 26255 in the event-only catalog. Note that the event-station catalog is a subset of the event-only catalog.

We list our true positive detection rates for four different MP thresholds and for different event-station weighted events in Table 5.4. In general, for the event-station catalog we detect 94 percent of all events and 98 percent of the W0 events using a threshold of 0.8. For the thresholds of 0.95, 0.9 and 0.85 we have a true positive rate of 76, 89 and 95 percent for strong detected events (weighed 0). This indicates that we had a very high true positive rates with respect to the event-station catalog. Figure 5.10 shows an example of a detected event waveform and the predicted MP for that event.

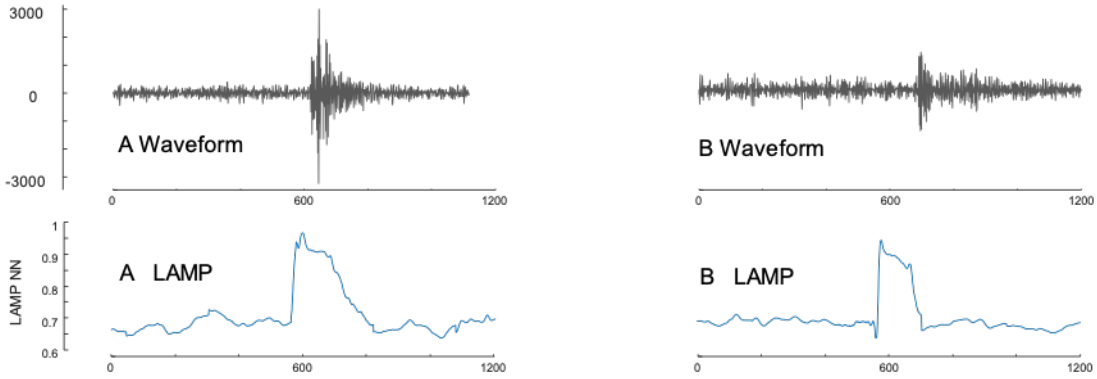

Figure 5.10: a) Example of an event from the event-station catalog detected by LAMP. b) Example of an event detected by LAMP that was not in the event-station catalog but was in the event catalog.

One interesting thing that we observe by experimenting with LAMP on this data set is that when using the 0.9 threshold we detect 1962 events from the event-only catalog that are not in the event-station catalog. This could be because these events occur far from the station, and thus produce weak seismic signals that a human analyst or the STA/LTA method could not identify, but have sufficiently similar characteristics to other events that LAMP could identify. Figure 5.10.b is one example of such an event.

After removing these ∼2k events plus the true positives from the event-station catalog, we end up with 48454 detected motifs that are not associated with any catalog events (i.e. not in either catalog; our "false" positives from Table 5.4). By visually inspecting these detected motifs, we group them into four categories:

(i) Earthquake waveforms for events missed by the catalog (Figure 5.11.a).

(ii) Station glitches (Figure 5.11.b), which can be caused by voltage surges or the electromagnetic radiation from a lightning strike.

(iii) Station artifacts, such as internal instrument calibration pulses (Figure 5.11.c).

(iv) Harmonic noise, possibly related to human activity or surface activity (Figure 5.11.d). For example, a gust of wind or earthmoving equipment.

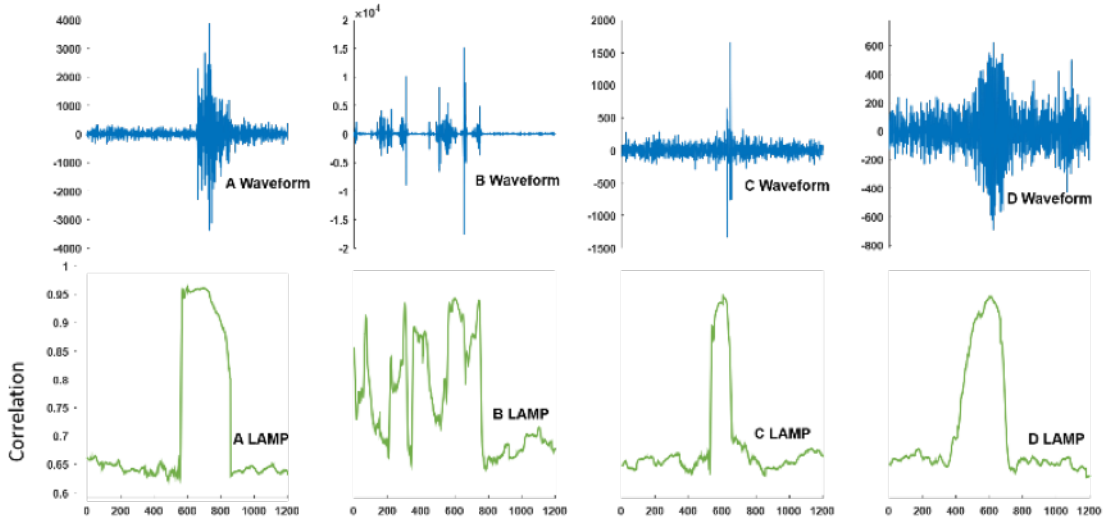

Figure 5.11: Examples of various non-catalog events detected by LAMP. a) Earthquake not in any catalog b) Station glitches c) Station artifacts d) Harmonic noise.

Clearly type (i) is the most exciting for seismologists, allowing them to populate their models and catalogs with additional examples that are currently missing.

Type (ii) and (iii) motifs can be easily removed from the data set by applying a simple query search using one of these instrumental errors as a query. Note that future LAMP models could be trained to ignore those signals. Type (iv) motifs can potentially be investigated by using LAMP on several stations to constrain their locations, which may be diagnostic of the source (e.g. a source located in the ocean might be caused by ocean waves and storms; a source located at the land surface could be weather or human-mediated).

Approximately 5% of the motifs discovered do not fit into this classification and are currently being investigated.

5.4.3 Case Study in Entomology

Across the world, there are hundreds of species of insects that feed by ingesting plant fluids. Some of these insects can cause damage to their host plants by transmitting pathogens. As a concrete example, the Asian citrus psyllid (Diaphorina citri) shown in Figure 5.12.left is a vector of the pathogen causing huánglóngbìng (citrus greening disease), which has already caused billions of dollars of damage to Florida's citrus industry in recent years, and is poised to do more damage worldwide. To design effective interventions, entomologists worldwide are attempting to understand the feeding behavior of such insects. As Figure 5.12 hints at, one of the most important tools used to study such insects is an EPG apparatus, which records the insects behavior as a one-dimensional time series [108].

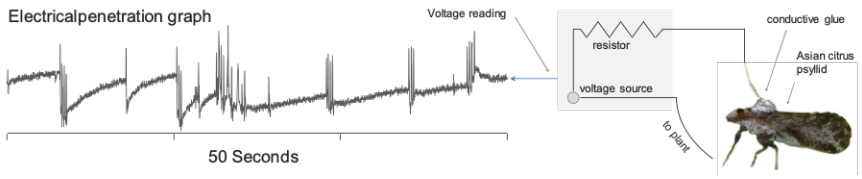

Figure 5.12: left) Fifty seconds of data collected from an EPG apparatus (center), which measures the changes in resistance as an insect interacts with a host plant. This data comes from a psyllid (right).

Dozens of labs worldwide collect such data, but to the best of our knowledge, all analytics are conducted in post-hoc batch sessions, missing the opportunity to test hypotheses in real time. For example, a recent paper suggests that the Asian citrus psyllid changes its behavior in response to some "combination of long and short wavelengths" [75]. Other research has suggested that various cocktails of volatile organic compounds can modify their behavior [6]. With such a huge search space of optical and semiochemicals parameters progress in designing interventions has been slow. Researchers have resorted to making a single change each session then adjusting the intervention for next day's session. However, if we could measure the behavior the insects in real-time, the entomologists could adaptively tune the optical and/or chemical mixture to optimize its effectiveness. Below we will show that LAMP makes this possible.

We consider a dataset of insect behavior that records an Asian citrus psyllid feeding on a Citrus natsudaidai (a type of orange). We took the first seven hours of data (2,500,000 datapoints), and using the annotations of [108], we created two classes:

Class A: Xylem Ingestion/Stylet Passage (181 min)

Class B: Non-Probing (235 min)

Note that as the data snippets shown in Figure 5.1 and Figure 5.12 hint at, the data here is very complex and noisy. Moreover, each class is polymorphic: Class A features two different stages of a feeding behavior and Class B is something of a "catch-all" [108]. For our testing data we consider 1,013 minutes of data, collected from the same insect in a later session. The class balance in that session happens to be almost equal, whereas the class balance in the training data is more skewed, at about five to one.

We can use a combination of multiple LAMP models to create a nearest neighbor classifier. Given a representative dataset of class A (R_A), another from class B (R_B), and stream of EPG data (S), we can train two separate LAMP models. The first LAMP model, M_A, is trained to approximate the RMP of S where matches can only occur in R_A. The second model, M_B, is trained to approximate the RMP of S where matches can only occur in R_B. Given a new subsequence I from S, we can produce the output of M_A and M_B when they are shown I. We can then use the class represented by the model that generated the largest value as the label for I. Table 5.5 shows the results for EPG classification across all models.

Table 5.5: Comparison of EPG Classification Results

Method	**Accuracy (%)**
Exact RMP	97.7
LAMP Diverse Motifs (K = 1600)	86.5
LAMP Neural Network	97.8
Direct Neural Network Classifier	99.2

Note that the neural network performs very slightly better than the exact RMP for the same task, but the difference is not statistically significant. Note also that we have trained a direct classifier using the same neural network used for LAMP but minimizing

the binary cross entropy of the predicted labels versus the true labels. As expected, this classifier performs better than a LAMP NN-classifier, as LAMP is not trained directly for classification, However, LAMP remains competitive.

Figure 5.13 shows how the accuracy of the diverse motifs method improves as K is increased. Note that the tradeoff between model size, efficiency, and accuracy is not always clear cut.

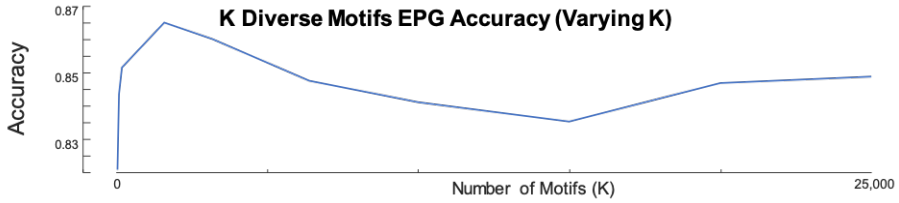

Figure 5.13: Effect on accuracy of varying the number of diverse motifs in the LAMP model.

5.4.4 Case Study on Pedestrian Traffic

The two previous case studies highlight the use of LAMP to predict high correlations, which are indicative of conserved structure. However, as we noted above, LAMP also predicts low correlations, which can be indicative of anomalies. To test the utility of LAMP in this context, we conducted the following experiment. As shown in Figure 5.14.top, we consider pedestrian traffic data from Bourke Street in Melbourne. We trained LAMP on 6.7 years of such data, beginning at 04/30/2009. For test data, we consider the following two years. While the test data surely has natural "anomalies" (usual weather/cultural events), to have some ground truth we embedded three synthetic anomalies:

- **Reversed**: A week of data was flipped backwards.

- **Replaced**: A week of data was replaced by a week of data from a different location in Melbourne (Southbank).

- **Diminished**: We simulated a sensor that slowly began to undercount over a week.

The first two anomalies are so subtle that they defy human inspection (Figure 5.14.top), and the third happens so slowly that examining only a few days at a time, it would be impossible to detect. Nevertheless, as Figure 5.14.bottom shows, a LAMP model with m = three days is able to correctly detect each of our three anomalies.

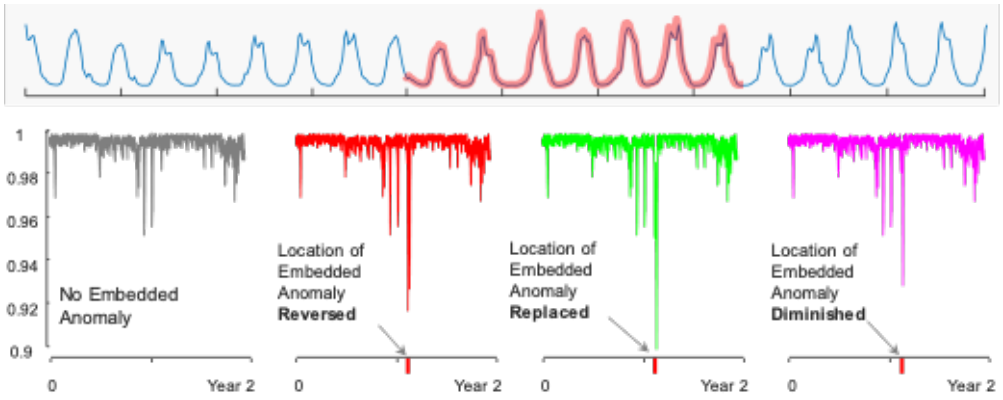

Figure 5.14: top) About 2% of the Reversed test dataset with embedded anomaly highlighted. bottom: left to right) The MP predicted by LAMP on two years of data with: no anomaly, the Reversed anomaly, the Replaced anomaly, and the Diminished anomaly. All three anomalous datasets have a significant dip in the LAMP output at the appropriate location.

5.4.5 When can LAMP fail?

The results above offer evidence that the LAMP framework can be useful in diverse settings, for diverse domains. Nevertheless, it is instructive to consider situations in which it can fail. LAMP very clearly can fail in the presence of concept drift [107], new motifs that have

never been seen before may arise from new underlying processes, and something that was a motif before may become an anomaly in the future (and vice-versa).

We defer a detailed discussion on retraining of LAMP to mitigate the effects of concept drift to Chapter 6 and future work. However, one simple way it can be done is to keep track of the last segment of observed data and use that to augment the representative dataset used to train LAMP. Every so often (or constantly in the background) we can retrain LAMP based on this augmented training data, and when the new model is ready, we can hot-swap the old model with the new and continue our inference.

5.5 Conclusion

We introduced LAMP, a flexible and generic framework that allow us to approximate the Matrix Profile values in the face of fast-moving streams. Because the Matrix Profile is at the heart of many time series algorithms for classification [90], motif discovery [109], anomaly detection [109], segmentation [37] etc., LAMP allows such higher-level algorithms to be used in real-time settings on fast moving streams that are currently untenable with the standard Matrix Profile.

Chapter 6

Future Work

6.1 Introduction and Disclaimer

This chapter introduces several future directions that researchers could take to extend the concepts introduced in this dissertation. Some of this work has been prototyped already, while other parts are speculative in nature. This chapter contains work which has not (as of the writing of this dissertation) been peer-reviewed. While we have made our best effort to present only facts in this chapter, the reader should assume that the claims presented in the following sections could be proven incorrect upon further experimentation.

6.2 Extending the SCAMP Framework

The following subsections contain a brief summary of future work related to extensions of the SCAMP framework (Chapter 4).

6.2.1 SCAMP Reduction Variants

The MP is a reduction of the columns of the Distance Matrix. It reports the minmum distance (Definition 4), or maximum correlation (Definition 7) of each column. We can trivially define other reductions as long as they can be computed similarly. For example, instead of taking the maximum correlation of each column, we could find the sum or product of the values. Additionally, we can specify a threshold such that we only evaluate computed correlations above a certain threshold. This enables finding the count/frequency of sequences which are highly correlated to the one in question. With some tuning, it might be possible to generate a distribution of the correlations of each column by hashing them into buckets.

These operations can be equivalently performed on the distance profiles (Definition 3) or correlation profiles (Definition 6). However, if we are computing the entire distance matrix, we can save a large amount of time by performing these reductions in-situ during the matrix profile computation.

6.2.2 All-Neighbors SCAMP

Another way the SCAMP framework can be extended is to to enable the return of multiple results per column of the distance matrix. Through clever use of GPU memory and atomic operations we can generate all matches between iterations of the inner loop of SCAMP (see Section 4.2.6).

A detailed description of the algorithms are left to future work. Essentially, we leave the highly optimized inner loop alone, maintaining its high performance, and append any values exceeding the threshold to a global array using atomic operations. If the number

of outputs is relatively small, there is almost no overhead in doing this, in fact it can be *faster* than the standard matrix profile computation. If there are a tremendous number of matches, then we can have performance degradation of up to ∼5-10x. This is still far faster than checking each subsequence's distance profile individually.

For performance reasons, we cannot actually return every match in the matrix. If one match occurs too close to another (approximately 500 subsequences apart, based on the amount of elements covered by one complete run of SCAMP's inner loop) then we will only return the nearer of those two matches. This means that we can potentially miss matches that are close together and this technique is an approximation of the true all-pairs result.

Using this clever technique, it is possible to generate a pooled or subsampled version of very large distance matrices that would otherwise be impossible to view directly, this is illustrated by Figure 6.1.

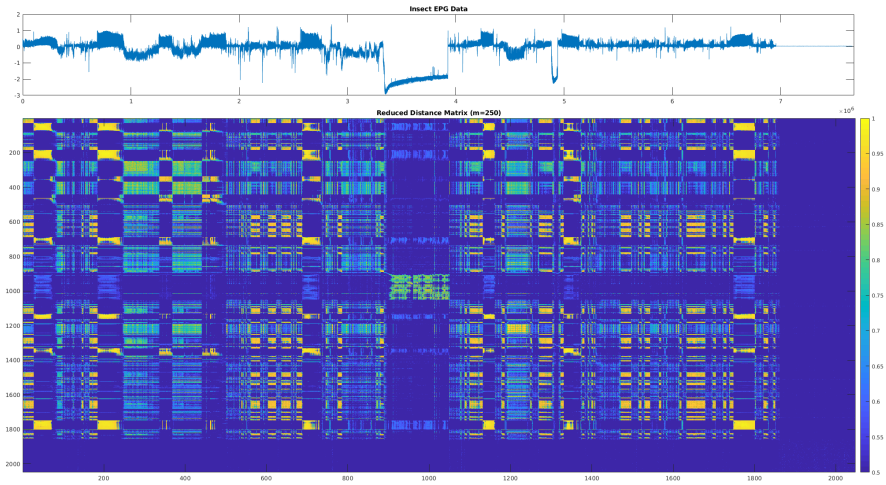

Figure 6.1: top) EPG Data from an Asian citrus psyllid [108]. bottom) Distance matrix showing implicit structure in the data. Each pixel represents the maximum correlation between groups of hundreds of consecutive subsequences.

These low resolution distance matrices could also be used by downstream clustering algorithms, for example spectral clustering [68] and biclustering [31], to unveil clusters that occur independently of the time domain (e.g. repeated motifs or regimes in the data).

Rather than using a matrix to represent the data, we can also use graph representations. Where each node represents a subsequence and edge between two nodes occurs when the two subsequences have a correlation above the specified threshold.

All-Neighbors SCAMP is compatible with the other reduction types mentioned in Section 6.2.1, for example we could return the motif frequency in the pooled matrix rather than pooling via the maximum correlation.

6.2.3 KNN SCAMP

It is also possible to remove the threshold parameter of All-Neighbors SCAMP and use an adaptive threshold, keeping track of the kth nearest neighbor distance of each column of the distance matrix. As SCAMP tiles are computed these thresholds can be dynamically updated so that by the end of SCAMP's execution, we have found the k nearest neighbors of each subsequence, of course this is only approximate based on the assumptions in the previous section. However, if each of the K nearest neighbors is sufficiently separated (500 subsequences or so apart) from the others, then this should produce the exact KNN.

6.3 Extending LAMP

The following subsections contain a brief summary of future directions that could be taken with LAMP (see Chapter 5).

6.3.1 Dynamic LAMP

From the end of Chapter 5, a major disadvantage of LAMP as described thus far is that it is not robust to any form of concept drift [107]. It is possible to improve this by building a tree structure, which catalogs historical segments of data, and can be used to selectively query various regions of historical data for matches. See Figure 6.2 for an illustration of this structure.

The leaf nodes of this tree structure model each historical segment of data that was observed. These segments can be checked by running predictions on new data with

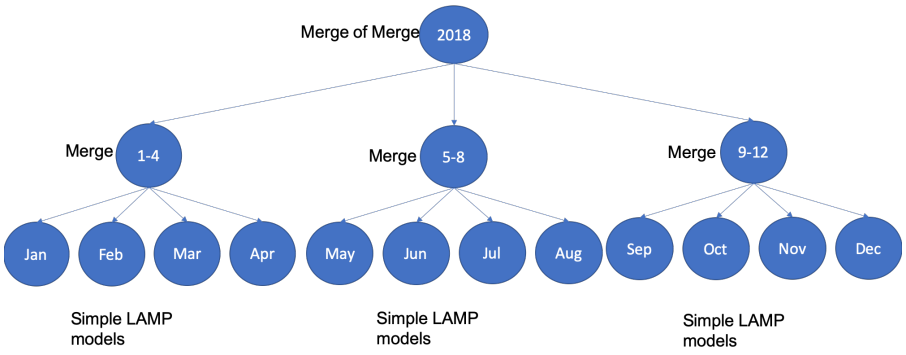

Figure 6.2: Illustration of a proposed LAMP tree structure. Enabling queries of data from variable regions in history and provides a rough version of the matrix profile index (see Definition 8) which can quantify how correlated a new segment of data is to any historical segment.

the leaf node's corresponding LAMP model. This LAMP model will output the expected correlation of this new data to subsequences in that old segment of data.

The non-leaf nodes of this tree structure can be trained by combining the predictions of their children nodes. In other words, a parent node is trained to predict the maximum correlation of any of its leaf nodes. This allows us to search the leaf nodes with fewer prediction steps, which allows faster predictions using the tree.

6.3.2 Dynamic LAMP Retraining

By itself, the tree structure defined in the previous section is relatively weak. It only catalogs a very sparse version of the distance matrix along the diagonal. However, we can retrain the leaf nodes of this structure for better performance, see Figure 6.3 for an illustration.

By performing the retraining as shown in Figure 6.3 we reduce the sparsity of the distance matrix represented by the tree. This will improve the results of future predictions

Figure 6.3: Illustration of a proposed LAMP tree retraining scheme, where leaf nodes are retrained via random selection. Green squares represent training which occurs during the normal tree building process, dark blue squares represent segments of the search space which are added via retraining.

as the distance matrix is better summarized. The amount of retraining performed can be set according to the user's computational resources and the incoming data sample rate.

6.3.3 Modeling other Similarity and Distance measures with LAMP

Using SCAMP, computing the Pearson Correlation Matrix is fast and efficient, we can perform a comparison between subsequences in $O(1)$. However there are other distance measures which are not so easy to compute. For example, dynamic time warping distance (DTW) [14] has a cost of $O(m^2)$, where m is the subsequence length. Is it possible to model DTW using LAMP? This is an exciting direction left to future work.

Chapter 7

Conclusions

The matrix profile enables the discovery of shape based information in time series. With insights from the matrix profile, data scientists and downstream algorithms can trivially perform motif discovery, discord discovery, segmentation and more.

Through expanding upon the scalability of MP construction we can enable deeper exploration into large time series datasets. This enhances the abilities data scientists and other downstream algorithms, allowing them to gain a complete picture of the shape based information in a dataset, even if it has hundreds of millions or billions of datapoints. As a case study, we show that these large matrix profiles can be extremely useful in seismology, where they can uncover many more events than exist in current seismic catalogs.

Using this highly scalable framework, we can generate enormous amounts of training data. Which allows us to push the boundaries of computation even further with a Learned Approximate Matrix Profile. With this advancement, we can utilize LAMP to compute an approximate matrix profile on streaming data from many sensors to generate

a picture of what is happening with our data in real-time. Even further, we can push the computation to the sensor, which enables active decision-making at the data source, rather than transmitting everything to the data center.

This work enables much more through its applications. Researchers can work with the matrix profile to tackle big datasets, and any of the use cases mentioned in this work could be applied uncover new knowledge in these other domains. There are many forms of time series data; only a minuscule fraction of them have been discussed in this work. The reader is encouraged to use any part of this work to enhance their own research and data mining techniques, and share their discoveries with the world.

Bibliography

[1] Geert Aarts, Monique MacKenzie, Bernie McConnell, Mike Fedak, and Jason Matthiopoulos. Estimating space-use and habitat preference from wildlife telemetry data. *Ecography*, 31(1):140–160, 2008.

[2] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In *International conference on foundations of data organization and algorithms*, pages 69–84. Springer, 1993.

[3] Keiiti Aki and Bernard Chouet. Origin of coda waves: source, attenuation, and scattering effects. *Journal of geophysical research*, 80(23):3322–3342, 1975.

[4] Richard M Allen, Holly Brown, Margaret Hellweg, Oleg Khainovski, Peter Lombard, and Douglas Neuhauser. Real-time earthquake detection and hazard assessment by elarms across california. *Geophysical Research Letters*, 36(5), 2009.

[5] Kate Allstadt and Stephen D Malone. Swarms of repeating stick-slip icequakes triggered by snow loading at mount rainier volcano. *Journal of Geophysical Research: Earth Surface*, 119(5):1180–1203, 2014.

[6] Berta Alquézar, Haroldo Xavier Linhares Volpe, Rodrigo Facchini Magnani, Marcelo Pedreira de Miranda, Mateus Almeida Santos, Nelson Arno Wulff, Jose Mauricio Simões Bento, José Roberto Postali Parra, Harro Bouwmeester, and Leandro Peña. β-caryophyllene emitted from a transgenic arabidopsis or chemical dispenser repels diaphorina citri, vector of candidatus liberibacters. *Scientific reports*, 7(1):5639, 2017.

[7] Amazon.com. Amazon ec2 spot instances. `https://aws.amazon.com/ec2/spot/`.

[8] Brian F Atwater, Alan R Nelson, John J Clague, Gary A Carver, David K Yamaguchi, Peter T Bobrowsky, Joanne Bourgeois, Mark E Darienzo, Wendy C Grant, Eileen Hemphill-Haley, et al. Summary of coastal geologic evidence for past great earthquakes at the cascadia subduction zone. *Earthquake spectra*, 11(1):1–18, 1995.

[9] Peter Bailis, Edward Gan, Kexin Rong, and Sahaana Suri. Prioritizing attention in fast data: Principles and promise. *CIDR Google Scholar*, 2017.

[10] William H Bakun and Allan G Lindh. The parkfield, california, earthquake prediction experiment. *Science*, 229(4714):619–624, 1985.

[11] Arvind Balasubramanian, Jun Wang, and Balakrishnan Prabhakaran. Discovering multidimensional motifs in physiological signals for personalized healthcare. *IEEE journal of selected topics in signal processing*, 10(5):832–841, 2016.

[12] Nurjahan Begum, Bing Hu, Thanawin Rakthanmanon, and Eamonn Keogh. Towards a minimum description length based stopping criterion for semi-supervised time series classification. In *2013 IEEE 14th International Conference on Information Reuse & Integration (IRI)*, pages 333–340. IEEE, 2013.

[13] Karianne J Bergen and Gregory C Beroza. Detecting earthquakes over a seismic network using single-station similarity measures. *Geophysical Journal International*, 213(3):1984–1998, 2018.

[14] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[15] Devin C Boyarko and Michael R Brudzinski. Spatial and temporal patterns of non-volcanic tremor along the southern cascadia subduction zone. *Journal of Geophysical Research: Solid Earth*, 115(B8), 2010.

[16] Emily E Brodsky. The importance of studying small earthquakes. *Science*, 364(6442):736–737, 2019.

[17] André EX Brown, Eviatar I Yemini, Laura J Grundy, Tadas Jucikas, and William R Schafer. A dictionary of behavioral motifs reveals clusters of genes affecting caenorhabditis elegans locomotion. *Proceedings of the National Academy of Sciences*, 110(2):791–796, 2013.

[18] Justin R Brown, Gregory C Beroza, and David R Shelly. An autocorrelation method to detect low frequency earthquakes within tremor. *Geophysical Research Letters*, 35(16), 2008.

[19] Diego Carrera, Beatrice Rossi, Pasqualina Fragneto, and Giacomo Boracchi. Domain adaptation for online ecg monitoring. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 775–780. IEEE, 2017.

[20] B Castello, M Olivieri, and G Selvaggi. Local and duration magnitude determination for the italian earthquake catalog, 1981–2002. *Bulletin of the Seismological Society of America*, 97(1B):128–139, 2007.

[21] Northern California Earthquake Data Center. Northern california seismic network catalog. `http://www.ncedc.org/ncsn/`.

[22] MJ Chae, HS Yoo, JY Kim, and MY Cho. Development of a wireless sensor network system for suspension bridge health monitoring. *Automation in Construction*, 21:237–252, 2012.

[23] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[24] Min Chen, Shiwen Mao, Yin Zhang, and Victor CM Leung. *Big Data: Related Technologies, Challenges and Future Prospects*. Springer, 2014.

[25] Bill Chiu, Eamonn Keogh, and Stefano Lonardi. Probabilistic discovery of time series motifs. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 493–498. ACM, 2003.

[26] Nvidia Corp. Fast histograms using shared atomics in maxwell. `https://devblogs.nvidia.com/gpu-pro-tip-fast-histograms-using-shared-atomics-maxwell/`.

[27] Nvidia Corp. Nvidia cuda c programming guide. `http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf`.

[28] Nvidia Corp. Nvidia cufft library user's guide. `http://docs.nvidia.com/cuda/pdf/CUFFT_Library.pdf`.

[29] Hoang Anh Dau and Eamonn Keogh. Matrix profile v: A generic technique to incorporate domain knowledge into motif discovery. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 125–134. ACM, 2017.

[30] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[31] Inderjit S Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–274. ACM, 2001.

[32] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. The lernaean hydra of data series similarity search: An experimental evaluation of the state of the art. *Proceedings of the VLDB Endowment*, 12(2):112–127, 2018.

[33] Bahaeddin Eravci and Hakan Ferhatosmanoglu. Diverse relevance feedback for time series with autoencoder based summarizations. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2298–2311, 2018.

[34] Edward H Field, Ramon J Arrowsmith, Glenn P Biasi, Peter Bird, Timothy E Dawson, Karen R Felzer, David D Jackson, Kaj M Johnson, Thomas H Jordan, Christopher Madden, et al. Uniform california earthquake rupture forecast, version 3 (ucerf3)—the time-independent model. *Bulletin of the Seismological Society of America*, 104(3):1122–1180, 2014.

[35] Yukio Fujinawa and Yoichi Noda. Japan's earthquake early warning system on 11 march 2011: performance, shortcomings, and changes. *Earthquake Spectra*, 29(s1):S341–S368, 2013.

[36] Robert J Geller and Charles S Mueller. Four similar earthquakes in central california. *Geophysical Research Letters*, 7(10):821–824, 1980.

[37] Shaghayegh Gharghabi, Chin-Chia Michael Yeh, Yifei Ding, Wei Ding, Paul Hibbing, Samuel LaMunion, Andrew Kaplan, Scott E Crouter, and Eamonn Keogh. Domain agnostic online semantic segmentation for multi-dimensional time series. *Data mining and knowledge discovery*, 33(1):96–130, 2019.

[38] Howard B Glasgow, JoAnn M Burkholder, Robert E Reed, Alan J Lewitus, and Joseph E Kleinman. Real-time remote monitoring of water quality: a review of current applications, and advancements in sensor, telemetry, and computing technologies. *Journal of experimental marine biology and ecology*, 300(1-2):409–448, 2004.

[39] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, 2018.

[40] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

[41] G Gutman and A1 Ignatov. The derivation of the green vegetation fraction from noaa/avhrr data for use in numerical weather prediction models. *International Journal of remote sensing*, 19(8):1533–1543, 1998.

[42] Sebastian Hainzl and David Marsan. Dependence of the omori-utsu law parameters on main shock magnitude: Observations and modeling. *Journal of Geophysical Research: Solid Earth*, 113(B10), 2008.

[43] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[44] Mark Harris et al. Optimizing parallel reduction in cuda. *Nvidia developer technology*, 2(4):70, 2007.

[45] Jens Havskov and Gerardo Alguacil. *Instrumentation in earthquake seismology*, volume 358. Springer, 2004.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[47] Nhut-Minh Ho and Weng-Fai Wong. Exploiting half precision arithmetic in nvidia gpus. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 2017.

[48] Alexander R Hutko, Manochehr Bahavar, Chad Trabant, Robert T Weekly, Mick Van Fossen, and Timothy Ahern. Data products at the iris-dmc: Growth and usage. *Seismological Research Letters*, 88(3):892–903, 2017.

[49] Toshihiro Igarashi, Toru Matsuzawa, and Akira Hasegawa. Repeating earthquakes and interplate aseismic slip in the northeastern japan subduction zone. *Journal of Geophysical Research: Solid Earth*, 108(B5), 2003.

[50] Richard M Iverson, Daniel Dzurisin, Cynthia A Gardner, Terrence M Gerlach, Richard G LaHusen, Michael Lisowski, Jon J Major, Stephen D Malone, James A Messerich, Seth C Moran, et al. Dynamics of seismogenic volcanic extrusion at mount st helens in 2004–05. *Nature*, 444(7118):439, 2006.

[51] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[52] David B Kirk and W Hwu Wen-Mei. *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann, 2016.

[53] Fred W Klein. User's guide to hypoinverse-2000, a fortran program to solve for earthquake locations and magnitudes. Technical report, US Geological Survey, 2002.

[54] Berkeley Seismological Laboratory. High resolution seismic network, 2014.

[55] Yuhong Li, Man Lung Yiu, Zhiguo Gong, et al. Quick-motif: An efficient and scalable framework for exact motif discovery. In *2015 IEEE 31st International Conference on Data Engineering*, pages 579–590. IEEE, 2015.

[56] Li-Chun Lin and Han-Sheng Chuang. Analyzing the locomotory gaitprint of caenorhabditis elegans on the basis of empirical mode decomposition. *PloS one*, 12(7):e0181469, 2017.

[57] Wuman Luo, Haoyu Tan, Huajian Mao, and Lionel M Ni. Efficient similarity joins on massive high-dimensional datasets using mapreduce. In *2012 IEEE 13th International Conference on Mobile Data Management*, pages 1–10. IEEE, 2012.

[58] Mohsen Marjani, Fariza Nasaruddin, Abdullah Gani, Ahmad Karim, Ibrahim Abaker Targio Hashem, Aisha Siddiqa, and Ibrar Yaqoob. Big iot data analytics: architecture, opportunities, and open research challenges. *IEEE Access*, 5:5247–5261, 2017.

[59] K. Mauck. Personal communication. 2018.

[60] Amy McGovern, Derek H Rosendahl, Rodger A Brown, and Kelvin K Droegemeier. Identifying predictive multi-dimensional time series motifs: an application to severe weather prediction. *Data Mining and Knowledge Discovery*, 22(1-2):232–258, 2011.

[61] Xiaofeng Meng, Xiao Yu, Zhigang Peng, and Bo Hong. Detecting earthquakes around salton sea following the 2010 mw7. 2 el mayor-cucapah earthquake using gpu parallel computing. *Procedia Computer Science*, 9:937–946, 2012.

[62] Rob van der Meulen. Gartner says 8.4 billion connected "things" will be in use in 2017 up 31 percent from 2016. *Gartner. Letzte Aktualisierung*, 7:2017, 2017.

[63] David Minnen, Charles L Isbell, Irfan Essa, and Thad Starner. Discovering multivariate motifs using subsequence density estimation and greedy mixture learning. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 615. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[64] Sarah E Minson, Men-Andrin Meier, Annemarie S Baltay, Thomas C Hanks, and Elizabeth S Cochran. The limits of earthquake early warning: Timeliness of ground motion estimates. *Science advances*, 4(3):eaaq0504, 2018.

[65] Abdullah Mueen, Eamonn Keogh, Qiang Zhu, Sydney Cash, and Brandon Westover. Exact discovery of time series motifs. In *Proceedings of the 2009 SIAM international conference on data mining*, pages 473–484. SIAM, 2009.

[66] Randall Munroe. Google's datacenters on punch cards. `https://what-if.xkcd.com/63/`.

[67] Robert M Nadeau, W Foxall, and TV McEvilly. Clustering and periodic recurrence of microearthquakes on the san andreas fault at parkfield, california. *Science*, 267(5197):503–507, 1995.

[68] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[69] SEJ Nippress, A Rietbrock, and AE Heath. Optimized automatic pickers: application to the ancorp data set. *Geophysical Journal International*, 181(2):911–925, 2010.

[70] Tesla NVIDIA. V100 gpu architecture, 2017.

[71] Kazushige Obara and Aitaro Kato. Connecting slow earthquakes to huge earthquakes. *Science*, 353(6296):253–257, 2016.

[72] Takahiro Omi, Yosihiko Ogata, Yoshito Hirata, and Kazuyuki Aihara. Forecasting large aftershocks within one day after the main shock. *Scientific reports*, 3:2218, 2013.

[73] Emmanuel Oyekanlu. Predictive edge computing for time series of industrial iot and large scale critical infrastructure based on open-source software analytic of big data. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1663–1669. IEEE, 2017.

[74] Stavros Papadopoulos, Kushal Datta, Samuel Madden, and Timothy Mattson. The tiledb array data storage manager. *Proceedings of the VLDB Endowment*, 10(4):349–360, 2016.

[75] Thomson M Paris, Sandra A Allan, Bradley J Udell, and Philip A Stansly. Evidence of behavior-based utilization by the asian citrus psyllid of a combination of uv and green or yellow wavelengths. *PloS one*, 12(12):e0189228, 2017.

[76] Zhigang Peng and Peng Zhao. Migration of early aftershocks following the 2004 parkfield earthquake. *Nature Geoscience*, 2(12):877, 2009.

[77] G Poupinet, WL Ellsworth, and J Frechet. Monitoring velocity variations in the crust using earthquake doublets: An application to the calaveras fault, california. *Journal of Geophysical Research: Solid Earth*, 89(B7):5719–5731, 1984.

[78] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 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, 2013.

[79] Kexin Rong and Peter Bailis. Asap: Automatic smoothing for attention prioritization in streaming time series visualization. *arXiv preprint arXiv:1703.00983*, 2017.

[80] Kexin Rong, Clara E Yoon, Karianne J Bergen, Hashem Elezabi, Peter Bailis, Philip Levis, and Gregory C Beroza. Locality-sensitive hashing for earthquake detection: A case study of scaling data-driven science. *Proceedings of the VLDB Endowment*, 11(11):1674–1687, 2018.

[81] Zachary E Ross and Yehuda Ben-Zion. Automatic picking of direct p, s seismic phases and fault zone head waves. *Geophysical Journal International*, 199(1):368–381, 2014.

[82] Zachary E Ross, Men-Andrin Meier, and Egill Hauksson. P wave arrival picking and first-motion polarity determination with deep learning. *Journal of Geophysical Research: Solid Earth*, 123(6):5120–5129, 2018.

[83] Zachary E Ross, Daniel T Trugman, Egill Hauksson, and Peter M Shearer. Searching for hidden earthquakes in southern california. *Science*, 364(6442):767–771, 2019.

[84] AA Royer and MG Bostock. A comparative study of low frequency earthquake templates in northern cascadia. *Earth and Planetary Science Letters*, 402:247–256, 2014.

[85] Justin L Rubinstein, David R Shelly, and William L Ellsworth. Non-volcanic tremor: A window into the roots of fault zones. In *New Frontiers in Integrated Solid Earth Sciences*, pages 287–314. Springer, 2009.

[86] WRM Sandanayaka, Y Jia, and JG Charles. Epg technique as a tool to reveal host plant acceptance by xylem sap-feeding insects. *Journal of Applied Entomology*, 137(7):519–529, 2013.

[87] Claudio Satriano, Anthony Lomax, and Aldo Zollo. Real-time evolutionary earthquake location for seismic early warning. *Bulletin of the Seismological Society of America*, 98(3):1482–1494, 2008.

[88] David P Schaff and Felix Waldhauser. Waveform cross-correlation-based differential travel-time measurements at the northern california seismic network. *Bulletin of the Seismological Society of America*, 95(6):2446–2461, 2005.

[89] David P Schaff and Felix Waldhauser. One magnitude unit reduction in detection threshold by cross correlation applied to parkfield (california) and china seismicity. *Bulletin of the Seismological Society of America*, 100(6):3224–3238, 2010.

[90] Nader Shakibay Senobari, Gareth J Funning, Eamonn Keogh, Yan Zhu, Chin-Chia Michael Yeh, Zachary Zimmerman, and Abdullah Mueen. Super-efficient cross-correlation (sec-c): A fast matched filtering code suitable for desktop computers. *Seismological Research Letters*, 90(1):322–334, 2018.

[91] BK Sharma, Amod Kumar, and VM Murthy. Evaluation of seismic events detection algorithms. *Journal of the Geological Society of India*, 75(3):533–538, 2010.

[92] David R Shelly. Migrating tremors illuminate complex deformation beneath the seismogenic san andreas fault. *Nature*, 463(7281):648, 2010.

[93] David R Shelly, Gregory C Beroza, and Satoshi Ide. Non-volcanic tremor and low-frequency earthquake swarms. *Nature*, 446(7133):305, 2007.

[94] David R Shelly, Gregory C Beroza, Satoshi Ide, and Sho Nakamula. Low-frequency earthquakes in shikoku, japan, and their relationship to episodic tremor and slip. *Nature*, 442(7099):188, 2006.

[95] David R Shelly, William L Ellsworth, Trond Ryberg, Christian Haberland, Gary S Fuis, Janice Murphy, Robert M Nadeau, and Roland Bürgmann. Precise location of san andreas fault tremors near cholame, california using seismometer clusters: Slip on the deep extension of the fault? *Geophysical research letters*, 36(1), 2009.

[96] Alejandro Simeone and Rory P Wilson. In-depth studies of magellanic penguin (spheniscus magellanicus) foraging: can we estimate prey consumption by perturbations in the dive profile? *Marine Biology*, 143(4):825–831, 2003.

[97] R Steve J Sparks. Forecasting volcanic eruptions. *Earth and Planetary Science Letters*, 210(1-2):1–15, 2003.

[98] Yoshiki Tanaka, Kazuhisa Iwamoto, and Kuniaki Uehara. Discovery of time-series motif from multi-dimensional data based on mdl principle. *Machine Learning*, 58(2-3):269–300, 2005.

[99] Quick Motif Team. Quickmotif webpage. `http://degroup.cis.umac.mo/quickmotifs/`,.

[100] UCR Matrix Profile Team. Matrix Profile Project Webpage. `http://www.cs.ucr.edu/eamonn/matrixprofile.html`.

[101] Tokuji Utsu, Yosihiko Ogata, et al. The centenary of the omori formula for a decay law of aftershock activity. *Journal of Physics of the Earth*, 43(1):1–33, 1995.

[102] Alireza Vahdatpour, Navid Amini, and Majid Sarrafzadeh. Toward unsupervised activity discovery using multi dimensional motif detection in time series. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[103] Radu-Daniel Vatavu. Small gestures go a long way: how many bits per gesture do recognizers actually need? In *Proceedings of the Designing Interactive Systems Conference*, pages 328–337. ACM, 2012.

[104] Lei Wang, Eng Siong Chng, and Haizhou Li. A tree-construction search approach for multivariate time series motifs discovery. *Pattern Recognition Letters*, 31(9):869–875, 2010.

[105] Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309, 2013.

[106] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 international joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[107] Geoffrey I Webb, Loong Kuan Lee, François Petitjean, and Bart Goethals. Understanding concept drift. *arXiv preprint arXiv:1704.00362*, 2017.

[108] Denis S Willett, Justin George, Nora S Willett, Lukasz L Stelinski, and Stephen L Lapointe. Machine learning for characterization of insect vector feeding. *PLoS computational biology*, 12(11):e1005158, 2016.

[109] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 1317–1322. IEEE, 2016.

[110] Clara E Yoon, Ossian O'Reilly, Karianne J Bergen, and Gregory C Beroza. Earthquake detection through computationally efficient similarity search. *Science advances*, 1(11):e1501057, 2015.

[111] Weiqiang Zhu and Gregory C Beroza. Phasenet: a deep-neural-network-based seismic arrival-time picking method. *Geophysical Journal International*, 216(1):261–273, 2018.

[112] Yan Zhu, Makoto Imamura, Daniel Nikovski, and Eamonn Keogh. Introducing time series chains: a new primitive for time series data mining. *Knowledge and Information Systems*, 60(2):1135–1161, 2019.

[113] Z. Zimmerman. Lamp supporting webpage. `https://sites.google.com/view/lamp2019`.

[114] Z. Zimmerman. Scamp supporting webpage. `https://sites.google.com/view/2019scamp`.