Journal of Computational Science 63 (2022) 101784

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

TraTSA: A Transprecision Framework for Efficient Time Series Analysis

Ivan Fernandez *, Ricardo Quislant, Sonia Gonzalez-Navarro, Eladio Gutierrez, Oscar Plata

Department of Computer Architecture, University of Malaga, Spain

ARTICLE INFO

Keywords:
Time series analysis
Transprecision Computing
Floating-Point Unit
FPGA
Parallel architectures

ABSTRACT

Time series analysis (TSA) comprises methods for extracting information in domains as diverse as medicine, seismology, speech recognition and economics. *Matrix Profile* (MP) is the state-of-the-art TSA technique, which provides the most similar neighbor to each subsequence of the time series. However, this computation requires a huge amount of floating-point (FP) operations, which are a major contributor (≈ 50%) to the energy consumption in modern computing platforms. In this sense, *Transprecision Computing* has recently emerged as a promising approach to improve energy efficiency and performance by using fewer bits in FP operations while providing accurate results.

In this work, we present *TraTSA*, the first transprecision framework for efficient time series analysis based on MP. TraTSA allows the user to deploy a high-performance and energy-efficient computing solution with the exact precision required by the TSA application. To this end, we first propose implementations of TraTSA for both commodity CPU and FPGA platforms. Second, we propose an accuracy metric to compare the results with the double-precision MP. Third, we study MP's accuracy when using a transprecision approach. Finally, our evaluation shows that, while obtaining results accurate enough, the FPGA transprecision MP (i) is 22.75× faster than a 72-core server, and (ii) the energy consumption is up to 3.3× lower than the double-precision executions.

1. Introduction

A time series is an ordered set of samples of a variable collected over time. It can contain millions of observations, often real-valued. Time series analysis seeks to extract information in a large variety of domains such as epidemiology [1], DNA analysis [2], economics [3], speech recognition [4], traffic prediction [5], energy conservation [6], seismology [7], medicine [8] and many more [9]. To do so, a common problem to solve is the all-pairs-similarity-search problem (also known as similarity join). That is, given a time series sliced in subsequences, retrieving the most similar and dissimilar subsequences. Particularly, *motif* [10] (similarity) and *discord* [11] (anomaly) discovery are two of the most frequently used primitives in time series data mining [12–18].

The state-of-the-art method for motif and discord discovery is *Matrix Profile* [19]. This method solves the similarity join problem and allows time-manageable computation of very large time series datasets. The similarity join has many applications [20] such as community discovery, duplicate text detection, collaborative filtering for social networks, clustering, and query refinement for web search.

In this work, we focus on this technique, which provides full joins without the need to specify a similarity threshold, a very challenging task in this domain. When analyzing a time series, the matrix profile is another time series representing the minimum distance subsequence for each subsequence (motifs). In contrast, maximum values of the profile highlight the most dissimilar subsequences (discords).

We evaluate the latest Euclidean-based implementations of matrix profile (SCRIMP [21] and SCAMP [22]) and find that a huge number of floating-point (FP) arithmetic operations are needed in order to analyze even short time series. In this sense, *transprecision computing* [23] has recently emerged as a promising approach to (i) improve energy efficiency, (ii) provide better performance, (iii) reduce area footprint, and (iv) reduce memory bandwidth by tolerating some loss of accuracy in computed results. This paradigm reduces the number of bits for the exponent and the mantissa in FP operations in a flexible way, depending on the requirements of the application. It is well known that FP operations are a major contributor (≈50%) [24] to the energy consumption in modern computing platforms. Thus, transprecision has the potential to provide an efficient design with the required precision by the application.

Our goal in this work is to provide a set of tools to jumpstart the research niche of transprecision time series analysis, to achieve high-performance and energy-efficient computing for a wide range of applications. This way, new platforms can be designed that benefit from

* Corresponding author.
E-mail addresses: ivanfv@uma.es (I. Fernandez), quislant@uma.es (R. Quislant), sgn@uma.es (S. Gonzalez-Navarro), eladio@uma.es (E. Gutierrez), oplata@uma.es (O. Plata).

https://doi.org/10.1016/j.jocs.2022.101784
Received 17 September 2021; Received in revised form 3 July 2022; Accepted 9 July 2022
Available online 20 July 2022

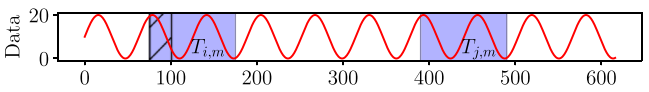

Fig. 1. Example of two subsequences, $T_{i,m}$ and $T_{j,m}$, of a given time series. When computing the matrix profile P, subsequences starting in the exclusion zone of $T_{i,m}$ are ignored for their high similarity.

reduced-bit-count FP operations tailored to each application (e.g., using a transprecision Floating-Point-Unit (FPU), like FPNew [25]). This opens up the opportunity to detect important events on mobile and embedded devices, where energy is a critical concern. Those devices can be used, for example, to prevent ecological disasters or medical issues (e.g., for early earthquake detection [26] or to predict a heart attack [27]).

To this end, we introduce *TraTSA*, the *first* transprecision framework for time series analysis. TraTSA, which is open-source and freely available to the community [28], provides fast and user-friendly transprecision matrix profile computing thanks to its CPU and FPGA implementations. We evaluate TraTSA with use cases of real datasets from different domains and sizes, analyzing the trade-offs between arithmetic precision and result accuracy using a proposed metric. Additionally, we present the energy savings of a real transprecision FPU.

The contributions of this work are the following:

- We develop TraTSA, the first framework for flexible transprecision matrix profile computation, which includes three transprecision implementations (`TranSCRIMP`, `TranSCAMP` and `TranSCAMPfpga`) based on FlexFloat [29] and cpfp-FPGA [30] libraries. We integrate those implementations using a Python wrapper that provides a user-friendly interface.
- We propose a new metric, called *Top-K Accuracy*, to measure the accuracy in the discovery of motifs and discords of a transformed time series (e.g., lower precision) with respect to the original time series.
- We provide a detailed accuracy analysis of `TranSCRIMP`, `TranSCAMP` and `TranSCAMPfpga` using the *Top-K Accuracy* metric while varying exponent and mantissa bit count combinations.
- We evaluate the potential benefits of using a transprecision FPU for matrix profile, finding that energy efficiency can be improved up to 3.3× compared with double precision with proper precision adjustments, while providing results accurate enough.

2. Background

In this section, we provide a background on time series analysis based on matrix profile, transprecision computing and FPGA acceleration.

2.1. Time series analysis. The matrix profile

A *time series* T is a sequence of n consecutive data points t_i, $1 \leq i \leq n$, collected over time. Time series data is typically stored using a floating-point representation. Let $T_{i,m}$ be a subsequence of T, where i is the index of its first data point, T_i, and m is the number of data points in the subset, with $1 \leq i$, and $m \leq n$. In the literature, $T_{i,m}$ is also called a *window* of length m. Fig. 1 shows an example of a sinusoidal time series with two subsequences highlighted, $T_{i,m}$ and $T_{j,m}$.

One way to measure the similarity between two time series subsequences is to use the *z-normalized Euclidean distance* [21], $d_{i,j}$, which is calculated as follows:

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

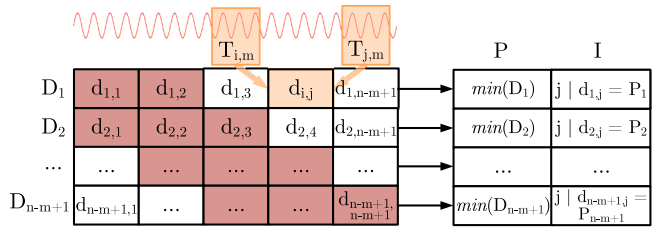

Fig. 2. Computation of the matrix profile P and the profile index I from the distance matrix D. P_i is the minimum distance of each row (or column) D_i. I_i is the index of the subsequence providing the minimum.

where $Q_{i,j}$ is the dot product of subsequences $T_{i,m}$ and $T_{j,m}$. μ_x and σ_x are the mean and the standard deviation of the data points in $T_{x,m}$, respectively.

Now we can use this distance measure to find the most similar subsequences out of all subsequences of a time series T. There are three steps to this procedure. First, building a symmetric $(n-m+1)\times(n-m+1)$ matrix D, called *distance matrix*, with a window size m and a time series of length n. Each D cell, $d_{i,j}$, stores the distance between two subsequences, $T_{i,m}$ and $T_{j,m}$. Thus each row (or column) of D stores the distances between a subsequence of T and every subsequence of T. Second, finding the two subsequences whose distance is minimum (i.e., most similar sequences) in each row (or column) of D. This can be computed by building the *matrix profile*, P, which is a vector of size $n - m + 1$. Each cell P_i in P stores the minimum value found in the ith row (or column) of D. Third, finding the indices of the most similar subsequences of the matrix profile. This requires building another vector I, called *matrix profile index*, of the same size as that of P, where $I_i = j$ if $d_{i,j} = P_i$. This way, P contains the minimum distances between subsequences of T while I is the vector of "pointers" to the location of these subsequences. Fig. 2 depicts an example of the distance matrix (D), the matrix profile (P), and the matrix profile index (I) for the time series in Fig. 1.

Notice that the D matrix is symmetric and the neighboring subsequences of $T_{i,m}$ are highly similar to it (i.e., $d_{i,i+1} \approx 0$) due to the overlaps. Thus, we can exclude these subsequences from computing D to find similar subsequences other than the neighboring ones. This is done by defining an exclusion zone (red cells in Fig. 2) for each subsequence. In general, the exclusion zone for $T_{i,m}$ is $\frac{m}{4}$ [21].

From the complexity point of view, matrix profile algorithms present two main challenges: memory footprint and a huge number of floating-point operations.

Memory footprint. The size of the distance matrix D can be huge for large time series, so it is not convenient to store it in memory. For example, an earthquake sequence from a seismograph, consisting of 24 000 elements [31], needs approximately 1.3 GB of memory using double-precision floating-point representation. However, a 650 000 electrocardiogram (ECG) time series from the MIT-BIH arrhythmia database [32] requires about 850 GB of memory. Furthermore, we need to maintain the matrix profile P, the matrix profile index I and the time series T. With the aim to overcome this issue, matrix profile algorithms are designed to store only the matrix profile and the matrix profile index arrays, computing the minimum distances $d_{i,j}$ on the fly.

Number of floating-point operations. The number of operations required even for short time series is the dominant part of the algorithm and increases quadratically with the time series length ($\approx 1.5 \times (n)^2$). Table 1 depicts the number of operations performed for a time series of 180k elements. As an example for this particular case, matrix profile needs to perform more than 64 billion multiplications. Thus, efficient floating-point units need to be designed to reduce the energy consumption and improve performance by increasing SIMD parallelism. Matrix profile implementations are usually based on double (64-bit) or single (32-bit) precision. The use of lower bit counts has not yet been well studied and is the main scope of this paper.

Table 1
Number of arithmetic floating-point operations performed in a time series of 180 000 elements and a window size of 512 elements.

Operation	Number	Operation	Number
Additions	16 084 915 120	Subtractions	183 664 640
Multiplications	64 340 019 200	FMAs	91 832 320
Inversions	16 084 915 120	Comparisons	32 170 906 916

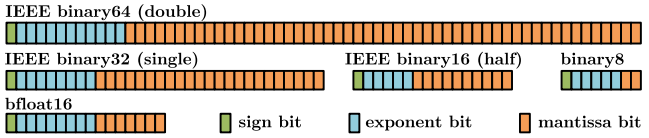

Fig. 3. Overview of the floating-point types used for energy evaluation.

Table 2
Floating-point bit counts, ranges and smallest numbers.

Type	Exp	Man	Range (≈)	Smallest (≈)
bin64	11	52	$\{-1.8 \cdot 10^{308}, 1.8 \cdot 10^{308}\}$	$2.2 \cdot 10^{-308}$
bin32	8	23	$\{-3.4 \cdot 10^{38}, 3.4 \cdot 10^{38}\}$	$1.2 \cdot 10^{-38}$
bin16	5	10	$\{-65\,504, 65\,504\}$	$6.1 \cdot 10^{-5}$
bfloat16	8	7	$\{-3.4 \cdot 10^{38}, 3.4 \cdot 10^{38}\}$	$1.2 \cdot 10^{-38}$
bin8	5	2	$\{-57\,344, 57\,344\}$	$6.1 \cdot 10^{-5}$

2.2. Transprecision computing

Transprecision Computing aims to boost energy efficiency and performance by exploiting numeric truncation in both hardware and software. It enables fine control over the precision of floating-point arithmetic in space and time (where and when to use it). The key difference with approximate computing is that transprecision guarantees the error (as the numeric precision and range are known for a given configuration) while approximation provides uncertainty. One example of this uncertainty occurs when reducing the refresh interval of RAM to improve performance [33], since process variation makes the exact error to be unknown. Based on that, transprecision leads to significant energy savings and performance improvements without sacrificing overall quality of results.

Transprecision computing can be applied to the entire algorithm by setting fixed exponent and mantissa widths for every floating-point operation. However, it is possible to change the precision to different parts of the code to find a trade-off between the accuracy of the results and energy efficiency. Mixed precision of double and single floating-point operations has been successfully used in the past [34,35] with a significant gain in performance.

Besides the IEEE-754 standard, transprecision computing allows the use of arbitrary exponent and mantissa bit combinations. However, the design of arbitrary precision floating-point units (FPUs) can be challenging and presents difficulties when integrating in computing platforms. Because of this reason, already designed FPUs typically support a fixed number of exponent and mantissa combinations. As an example, Fig. 3 shows the types [24] that we use in this work for the energy evaluation.

Table 2 summarizes the bit count for each floating-point datatype, along with the approximate range and smallest number that can be represented. Notice that, while *binary32* and *binary16* have approximately the same range, the first one provides narrower steps between numbers and more precision.

We find transprecision Floating-Point Units (FPU) already proposed in the literature, which aim to take advantage of transprecision computing in terms of energy, performance and area. The first silicon implementation of a 64-bit transprecision FPU can be found in [36], using 22 nm technology node. This FPU supports the floating-point types depicted in Fig. 3, and the key idea behind it is to operate in scalars (64-bit) or in SIMD vectors of 2 elements (32-bit), 4 elements (16-bit) or 8 elements (8-bit). The authors of such FPU evaluate it integrated into a RISC-V core via simulation (but can be used in FPGAs or ASICs) and obtain speedups up to 7.3× while reducing energy up to 7.94× with respect to *double* precision approaches. The source code of this FPU (known as FPnew) can be found in [25].

In contrast, transprecision libraries aim to enable transprecision software emulation in commodity architectures. An example of CPU-based transprecision emulation is FlexFloat [29], which is written in C. The main advantage of this library is that it can be executed in almost any commodity platform, but it presents a main drawback: the significant overhead introduced by the software emulation. In this sense, we observe that the FlexFloat execution time increases the native IEEE double execution time by 200× for a given computing platform. One way to overcome this issue is the use of an FPGA architecture, including custom-precision units from cpfp-FPGA [30] library, which is intended for High-Level Synthesis (HLS). While not being a pure transprecision implementation from the architectural point of view (e.g., data is stored using 32 bits in memory and then converted to the desired width), it allows time-manageable evaluation of large time series (millions of elements). The key motivation to consider the FPGA acceleration is to avoid the emulation cost of the software-based one with the use of dedicated transprecision hardware.

Table 3
Xilinx Alveo U50 Specifications.

Parameter	Alveo U50
Look-up Tables (LUTs)	872k
Registers	1743k
DSP Slices	5952
Memory	8 GB (HBM)
HBM Bandwidth	316 GB/s (theoretical)
Internal SRAM	28 MB
Int. SRAM Bandwidth	24 TB/s
Max. Power	75 W

2.3. FPGA acceleration

Field Programmable Gate Arrays (FPGAs) [37] are programmable devices comprising logic blocks which connection and behavior can be defined by the user. This kind of devices can be programmed to implement a wide variety of algorithms and even full-fledged systems, such as a complex RISC-V core [38].

The main advantage of using an FPGA to accelerate a given algorithm, compared with implementing a full system, is that more FPGA resources are available for the implementation of the required functionality. For instance, resources devoted to branch prediction or instruction decoding in the implementation of a general-purpose core can be used to place more floating-point units for the algorithm instead.

One example of an HPC FPGA board is the Xilinx Alveo U50. This PCIe board, which includes 3D-stacked High-Bandwidth Memory [39] HBM, is a good candidate to accelerate memory-bound applications such as time series analysis [40]. Concretely, the HBM cube in this board exposes 30 pseudochannels that enable memory-level parallelism. Applications can exploit this parallelism by (i) spanning multiple compute units (CUs) and (ii) allocating data structures in different pseudochannels. Because of that, we use it in the evaluation of this work. Table 3 shows the specifications of the U50.

3. TraTSA framework

In this section we present an overview of the TraTSA framework and then describe its main components.

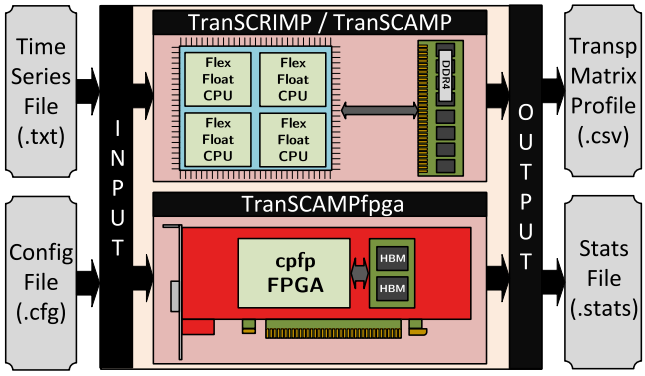

Fig. 4. TraTSA overview and its components. The user provides a time series file (.txt) and a configuration file (.cfg) to the wrapper. Then, the wrapper invokes matrix profile either in the CPU or in the FPGA. Finally, the wrapper provides the user the transprecision matrix profile (.csv) and some statistics (.stats).

example.cfg

```
7 16 # distance_exponent   distance_mantissa
7 16 # dotproduct_exponent dotproduct_mantissa
7 16 # statistics_exponent statistics_mantissa
7 16 # profile_exponent    profile_mantissa
1024 # window_size
0    # backend (0 = TranSCRIMP,
     #          1 = TranSCAMP,
     #          2 = TranSCAMPfpga)
```

Fig. 5. Example of TraTSA's cfg file.

3.1. Overview of TraTSA

TraTSA is a *Tra*nsprecision Framework for *T*ime *S*eries *A*nalysis developed as a tool (i) to perform design exploration of accelerators and (ii) to tune current implementations. This way, computer architects can define the exact number of floating-point bits for exponent and mantissa, which potentially saves area and improves performance while reducing energy consumption. We build TraTSA framework based on matrix profile and using (i) the FlexFloat library to implement transprecision CPU versions of SCRIMP and SCAMP, (ii) the cpfp-FPGA library to implement a transprecision FPGA implementation of SCAMP[1] and (iii) Python to create a user-friendly wrapper.

We present TraTSA's overview in Fig. 4. The *INPUT* and *OUTPUT* blocks in the figure represent TraTSA's wrapper.

This wrapper is in charge of (i) interpreting the configuration file (i.e., exponent and mantissa widths, selected CPU or FPGA backend, window size, among others) and obtaining the time series file provided by the user; (ii) invoking the corresponding execution backend, and (iii) collecting the results providing them to the user after proper formatting. We present a simple example of TraTSA's configuration file in Fig. 5, which is based on a custom format. The stats file follows a similar format to that provided by the FlexFloat [29] library.

TraTSA, being easily extensible to support additional algorithms, includes the following backends.

- **`TranSCRIMP`**. `TranSCRIMP` is a CPU C++ parallel transprecision implementation of SCRIMP algorithm. This implementation provides configurable precision arithmetic which is emulated via software using the FlexFloat library.
- **`TranSCAMP`**. `TranSCAMP` is a CPU C++ parallel transprecision implementation of SCAMP algorithm. This implementation provides configurable precision arithmetic which is emulated via software using the FlexFloat library.
- **`TranSCAMPfpga`**. `TranSCAMPfpga` is an FPGA HLS-based implementation of SCAMP algorithm. This implementation provides configurable precision arithmetic which is implemented via hardware using the cpfp-FPGA library. As we evaluate `TranSCAMPfpga` using a Xilinx Alveo U50 FPGA board, it is trivial to port it to other Alveo models.

The key benefit of TraTSA is to provide a transprecision framework being (i) portable enough to be executed in different execution environments according to the analysis requirements (i.e., length of the time series or if the user has access to an FPGA or not), and (ii) flexible enough to allow the possibility of exploring a wide range of exponent and mantissa combinations for any dataset. In this sense, both `TranSCRIMP` and `TranSCAMP` are designed to be used with time series of modest sizes (below 200k elements) and executed in commodity CPUs (desktops or high-end servers) due to the overheads of custom-precision types in those platforms. In contrast, `TranSCAMPfpga` is able to compute series of up to several million elements in manageable time thanks to the transprecision hardware support, at the cost of requiring an FPGA. Both CPU and FPGA backends can work simultaneously and join compute power.

3.2. Transprecision SCRIMP-CPU (`TranSCRIMP`)

`TranSCRIMP` is a CPU transprecision implementation of SCRIMP based on FlexFloat library. The key idea of SCRIMP [21] is to minimize the computation by exploiting the fact that the dot product can be updated incrementally for subsequences in the diagonal of the distance matrix, D (see Fig. 2). Consequently, the dot product can be expressed as follows:

$$Q_{i,j} = Q_{i-1,j-1} - T_{i-1}T_{j-1} + T_{i+m-1}T_{j+m-1} \tag{2}$$

The baseline implementation for our transprecision SCRIMP algorithm, `TranSCRIMP` in Fig. 6, is a vectorized-parallel version presented in [40]. It first precomputes the mean and standard deviation of each time series subsequence (line 1), and initializes the matrix profile array (line 3). Then, the distances between pairs of subsequences are calculated following the diagonals of the distance matrix (lines 4–26). The for loop is fully parallelized, with each thread computing a random subset of diagonals provided by their indices in the *diag* array in line 5.

For the first element of the diagonal, we need to compute the dot product of the first pair of subsequences (line 6) in parallel. The rest are updated following Eq. (2). For the proper vectorization of the dot product update, the algorithm separates the calculation of the diagonal in several steps: (i) the products in Eq. (2) are calculated in parallel for $vectFact$ elements of the diagonal (lines 14–15); (ii) the previous dot product, q, is added to the element calculated in step (i) (line 16); (iii) the subsequent dot products are updated sequentially using the previous ones (lines 17–18) saving the last one in q for the next iteration of the diagonal (line 19); (iv) distances are calculated in parallel (lines 20–21); and (v) the profile is updated in parallel as well (lines 22–25).

The vectorization factor, $vectFact$, is given by the transprecision datatype width with respect to that of double precision (line 2). We highlight the lines of code which are to be executed using the transprecision approach. The algorithm is able to work in either one precision configuration or a mixed-precision one, thus the green (lower precision) and red (higher precision) marks.

[1] We do not consider implementing a transprecision FPGA version of SCRIMP since SCAMP provides better numeric stability and, as a consequence, more robustness to reduced precision (see Section 5.2.1), being more amenable to transprecision approaches. However, due to the high-similar computation schemes of both algorithms, it is feasible to implement `TranSCRIMPfpga` with modest effort.

```
1:  μ, σ ← computeMeanDev(T, m);                          ▷ precalculation of statistics
2:  vectFact ← 8/sizeof(datatype);                        ▷ 8 bytes (double) by the size of the transprecision datatype
3:  P ← ∞;                                                ▷ matrix profile array initialization
4:  for idx ← tid * numDiag to (tid + 1) * numDiag − 1 do ▷ parallel for
5:      i ← 0; j ← diag_idx;                              ▷ diag array contains the diagonals assigned to each thread
6:      q ← dotProduct(T_i,m, T_j,m);                     ▷ compute dot product for the first diagonal element (SIMD)
7:      d ← dist(m, q, μ_i, σ_i, μ_j, σ_j);               ▷ compute distance for the first element of diagonal
8:      if d < P_i then                                   ▷ update matrix profile and index arrays
9:          P_i ← d; I_i ← j;
10:     if d < P_j then                                   ▷ distance matrix is symmetric
11:         P_j ← d; I_j ← i;
12:     i ← i + 1;
13:     for j ← diag_idx to size(P) do
14:         for k ← 0 to vectFact − 1 do                  ▷ compute new dot product elements in parallel (SIMD)
15:             qs_k ← t_i+m−1+k t_j+m−1+k − t_i−1+k t_j−1+k;
16:         qs_0 ← qs_0 + q                               ▷ update with the first dot product of the diagonal
17:         for k ← 1 to vectFact − 1 do                  ▷ update remaining dot products (sequentially)
18:             qs_k ← qs_k + qs_k−1;
19:         q ← qs_vectFact−1;                            ▷ store last dot product for next iteration
20:         for k ← 0 to vectFact − 1 do                  ▷ compute distances in parallel (SIMD)
21:             ds_k ← dist(m, qs_k, μ_i+k, σ_i+k, μ_j+k, σ_j+k);
22:             if ds_k < P_i+k then                      ▷ update matrix profile and index arrays
23:                 P_i+k ← ds_k; I_i+k ← j + k;
24:             if ds_k < P_j+k then
25:                 P_j+k ← ds_k; I_j+k ← i + k;
26:         i ← i + vectFact;
```

Fig. 6. Transprecision SCRIMP (`TranSCRIMP`) algorithm (transprecision operations highlighted).

Rather than using only double and single precision, we define a high and a low precision that can be set to any possible exponent and mantissa configuration, which provides further accuracy analysis opportunities. For the algorithms involved in this study, we store the time series codified with high precision and both the matrix profile and the precalculated statistics with low precision. For `TranSCRIMP`, Fig. 6 shows the lines of code that works with high precision highlighted in red, and those which work with low precision highlighted in green. The dot product calculations use high precision as they may require a larger numeric range. Distance calculation as well as calculations with means and standard deviations are performed with less precision.

3.3. Transprecision SCAMP-CPU (`TranSCAMP`)

`TranSCAMP` is a CPU transprecision implementation of SCAMP based on FlexFloat library. Whereas following a similar computation scheme to `TranSCRIMP`, `TranSCAMP` replaces the sliding dot product with a mean-centered-sum-of-products in order to reduce the floating-point rounding errors and the number of operations required [22]. The following equations can be precomputed in $O(n - m + 1)$ time, with $n - m + 1 = l$ being the length of P:

$$df_i = \frac{T_{i+m-1} - T_{i-1}}{2}, \quad 0 < i < l \tag{3}$$

$$dg_i = T_{i+m-1} - \mu_i + T_{i-1} - \mu_{i-1}, \quad 0 < i < l \tag{4}$$

$$ssq_i = \begin{cases} \sum_{k=0}^{m-1} (T_k - \mu_0)^2, & i = 0 \\ ssq_{i-1} + (T_{i+m-1} - \mu_i + \\ + T_{i-1} - \mu_{i-1})(T_{i+m-1} - T_{i-1}) & 0 < i < l \end{cases} \tag{5}$$

$$\sigma_i = \sqrt{ssq_i}, \quad 0 \le i < l \tag{6}$$

Eqs. (3) and (4) are terms used in the covariance update of Eq. (7), and the standard deviation (L2-norm of subsequence $T_{i,m} - \mu_i$) calculated in Eqs. (5) and (6) is used for the Pearson correlation coefficient depicted by Eq. (8). Notice the exclusion zone in the limits of Eq. (7) given by $\frac{m}{4}$.

$$\sigma_{i,j} = \begin{cases} \sum_{k=0}^{m-1} (T_k - \mu_0)(T_{k+j} - \mu_j), & i = 0, \frac{m}{4} < j < l \\ \sigma_{i-1,j-1} + df_i dg_j + df_j dg_i, & i > 0, \frac{m+4}{4} < j < l \end{cases} \tag{7}$$

$$P_{i,j} = \frac{\sigma_{i,j}}{\sigma_i \sigma_j} \tag{8}$$

$$D_{i,j} = \sqrt{2m(1 - P_{i,j})} \tag{9}$$

The matrix profile can be derived incrementally for each diagonal of the distance matrix, Eq. (7), from the calculation of the covariance of two subsequences of the first row (first piece in Eq. (7)). The Pearson correlation coefficient in Eq. (8) can be computed in fewer operations and it is more robust than the Euclidean Distance used by SCRIMP [22]. Eq. (9) calculates the distance from the Pearson coefficient in $O(1)$. For `TranSCAMP`, we compute the covariance in Eq. (7) at high precision, which may have a large numeric range depending on the series. The correlation in Eq. (8), which varies between −1 and 1, is computed at low precision.

3.4. Transprecision SCAMP-FPGA (`TranSCAMPfpga`)

`TranSCAMPfpga` is an FPGA transprecision implementation of SCAMP based on the cpfp-FPGA library. We include `TranSCAMPfpga` as part of TraTSA's backend to speed up the evaluation of transprecision time series analysis in those research environments where FPGAs are available. `TranSCAMPfpga` is tuned for a Xilinx Alveo U50 FPGA board, which includes High-Bandwidth-Memory (HBM) [41]. However, thanks to the C++-based HLS implementation, it is easy to port to higher-end Alveo boards or, with modest effort, to other FPGA platforms, as Intel Altera. We present an overview of `TranSCAMPfpga` in Fig. 7, including the kernels inside the two SLRs (Super Logic Regions) of the FPGA and the HBM memory.

We develop `TranSCAMPfpga` using Xilinx Vitis 2020.2 and a C++ HLS approach. Our implementation consists of (i) a host side code,

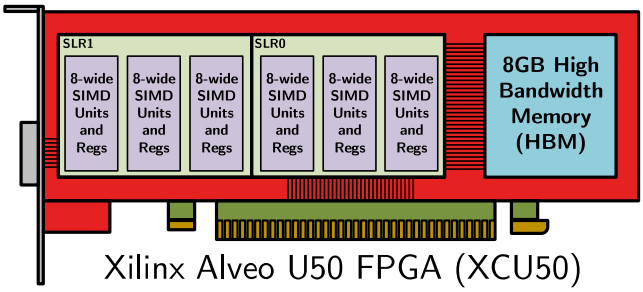

Fig. 7. FPGA implementation overview. `TranSCAMPfpga` is composed of six kernels optimized for a Xilinx Alveo U50 FPGA that compute transprecision SCAMP algorithm using the data in the HBM.

and (ii) an FPGA side code (kernels), that compute matrix profile in a parallel-vectorized manner. To avoid the use of synchronization primitives, each kernel has exclusive access to its private matrix profile.

On the one hand, the host code, which is executed in the host CPU, is in charge of (i) allocating the corresponding data structures in the FPGA and transferring the input vectors (i.e., time series and statistic data) from the host to the HBM cubes of the FPGA via PCI-Express, (ii) defining the parallel scheduling partitioning, (iii) invoking the FPGA, (iv) transferring the output vectors (i.e., matrix profiles) from the HBM cubes of the FPGA to the host via PCI-Express, and (v) performing the final matrix profile reduction.

On the other hand, the FPGA code is an HLS-based implementation that performs the transprecision computation of SCAMP. This FPGA implementation, intended for the evaluation of large time series (i.e., millions of elements), includes the possibility of defining a recalculation interval to reduce the accumulated errors across diagonals due to lower precision arithmetic. Concretely, instead of reusing the previous covariance to calculate the current one, it performs the complete centered-sum-of-products at certain interval. Regarding to performance, we tune this FPGA implementation to benefit as much as possible from the available resources of the FPGA, exploiting the following techniques:

- **Parallelization across kernels**. Instead of having a huge kernel that computes the whole matrix profile of the given time series, we create several kernels (six in the case of the Alveo U50) which allows performing diagonal-level parallelization. Additionally, the use of six kernels efficiently exploits the HBM memory as accesses can be overlapped.
- **Vectorization inside kernels**. Instead of calculating one diagonal at a time, each kernel computes a set of diagonals (512) which allows exploiting loop unrolling techniques via arithmetic operator replication, and reusing data via *systolic arrays*. Fig. 8 illustrates an example of the main advantage of vectorization using five diagonals. We notice that for the first elements of all diagonals, values of data structures need to be brought from the HBM memory. However, for the rest of the rows of all diagonals (as all diagonals increase i index at the same time), we only need to bring from memory one data value (the last one), regardless of the vectorization width. This can be achieved thanks to the systolic array approach, where previously fetched data are pushed forward to the next row while the new data value is read from memory at the same time. This operation is performed in parallel and takes only one step for all elements of the systolic array.
- **Wide memory accesses**. We pack data that is requested from memory to improve the efficiency of the bus (i.e., bringing 512 bits in each request) while performing burst accesses at the same time.

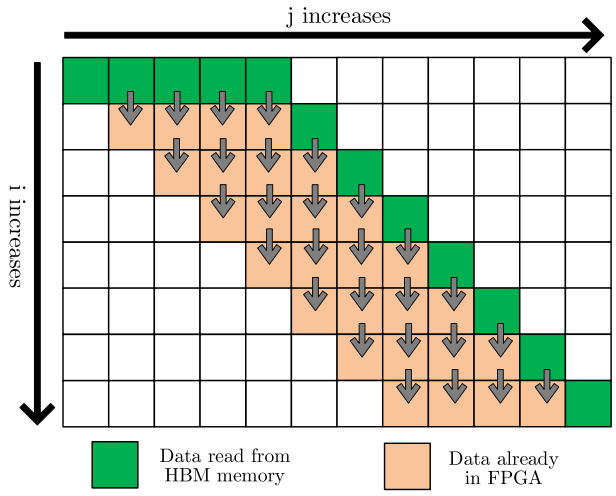

Fig. 8. `TranSCAMPfpga` systolic array example.

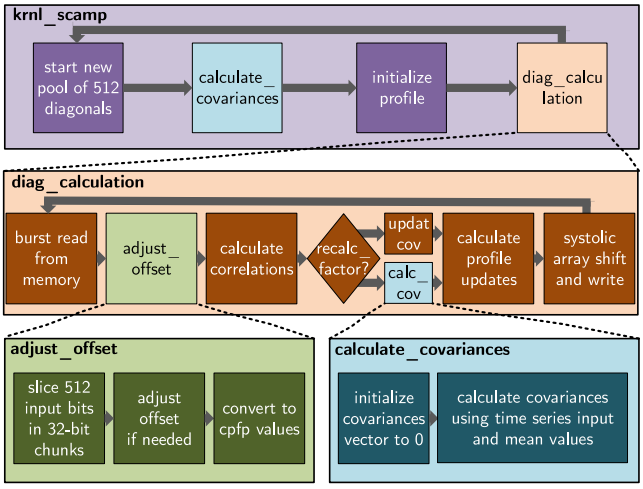

Fig. 9. FPGA implementation diagram.

Table 4
`TranSCAMPfpga` kernel resource utilization.

Param	Avail.	Used	Param	Avail.	Used
LUT	752 672	308 184 (41%)	URAM	640	0 (0%)
REG	1 586 939	494 199 (31%)	BRAM	1164	714 (61%)
LUTm	389 324	5802 (1.5%)	DSP	5936	828 (14%)

We present `TranSCAMPfpga`'s execution flowchart in Fig. 9, which comprises an external loop that iterates over diagonals in batches of 512, and an inner loop to go over those diagonals following a SIMD approach. Notice that the *calculate_covariances* module is shared among two steps of the execution, which allows to save hardware resources as it is not used in all iterations of the *diag_calculation* loop.

Finally, Table 4 shows the total resource utilization numbers by `TranSCAMPfpga` kernels in the Alveo U50 board. According to our observations using Vitis Profiler, the main bottleneck for the performance of this implementation is data movement. This means that increasing the number of arithmetic operations does not improve performance because (i) global clock frequency is reduced to enable proper functionality, and (ii) those additional arithmetic operations require increasing the number of memory ports, which leads to routing errors due to network congestion.

4. Top-K accuracy metric

Time series motifs [10] and discords [11] have been used for more than 15 years in the field of data mining for their capacity to find time series subsequences with special significance. In this section we define these special subsequences and propose a metric to measure the accuracy in the detection of motifs and discords from two time series.

The definitions of the motif and the Top-K motifs of a time series are presented in the remainder of this section.

Definition 1. The motif M_1 of a time series T is the unordered pair of subsequences $\{T_{i,m}, T_{j,m}\}$ which is the most similar among all possible pairs:

$$M_1 = \{T_{i,m}, T_{j,m}\} \Leftrightarrow dist(T_{i,m}, T_{j,m}) \leq dist(T_{u,m}, T_{v,m})$$

$$\forall i, j, u, v;\ i \neq j, u \neq v.$$

Definition 2. The Top-K motifs $M_{1,K}$ of a time series T is the set of the first K motifs:

$$M_{1,K} = \begin{cases} M_K \cup M_{1,K-1}, & K > 1 \\ M_1, & K = 1 \end{cases}$$

being M_K the motif (M_1) of the time series $T \setminus M_{1,K-1}, \forall K > 1$.

We can define the discord and the Top-K discords of a time series in a similar way:

Definition 3. The discord D_1 of a time series T is the unordered pair of subsequences $\{T_{i,m}, T_{j,m}\}$ which is the most dissimilar among all possible pairs:

$$D_1 = \{T_{i,m}, T_{j,m}\} \Leftrightarrow dist(T_{i,m}, T_{j,m}) \geq dist(T_{u,m}, T_{v,m})$$

$$\forall i, j, u, v;\ i \neq j, u \neq v.$$

Definition 4. The Top-K discords $D_{1,K}$ of a time series T is the set of the first K discords:

$$D_{1,K} = \begin{cases} D_K \cup D_{1,K-1}, & K > 1 \\ D_1, & K = 1 \end{cases}$$

being D_K the discord (D_1) of the series $T \setminus D_{1,K-1}, \forall K > 1$.

Being $MI_{1,K}$ and $DI_{1,K}$ the Top-K sets of unordered pair of indices $\{i, j\}$ of the unordered pair of subsequences in $M_{1,K}$ and $D_{1,K}$, respectively, we define the Top-K Motif/Discord Accuracy in the following way:

Definition 5. The Top-K Motif (Discord) Accuracy AM (AD) of a time series T with respect to another time series TT is the number of coincidences among the unordered pairs in $MI^{T}_{1,K}$ and $MI^{TT}_{1,K}$ ($DI^{T}_{1,K}$ and $DI^{TT}_{1,K}$):

$$AM^{T \to TT}_{1,K} = |MI^{T}_{1,K} \cap MI^{TT}_{1,K}|$$

$$AD^{T \to TT}_{1,K} = |DI^{T}_{1,K} \cap DI^{TT}_{1,K}|$$

Definition 5 is pointless when T and TT are time series from different applications. However, it can be useful when we want to know the degree of coincidence of the matrix profile of a time series and that of the same time series codified with less precision or affected by a transform operator. We can get to know if, after a transformation of the original time series, the matrix profile ends up unveiling the same motifs and discords, or a significant subset of them.

5. Experimental evaluation

5.1. Methodology

We evaluate TraTSA's `TranSCRIMP`, `TranSCAMP` and `TranSCAMPfpga` backends in terms of accuracy, performance and energy savings. First, we compute the reference SCRIMP and SCAMP matrix profiles with double and single floating-point precisions using the native C++ implementations. Second, we use the FlexFloat [29] library on an Intel Xeon Phi 7210 "Knights Landing" manycore processor [42] with 64 cores and 256 threads to evaluate `TranSCRIMP` and `TranSCAMP` on relatively short[2] time series (<200k elements). Third, we use the cpfp-FPGA library on a Xilinx Alveo U50 FPGA board to evaluate `TranSCAMPfpga` for larger time series (>200k elements). Fourth, we present performance comparisons between `TranSCAMP` and `TranSCAMPfpga` using different computing platforms. Finally, we compare the energy consumption in two ways. On the one hand, we study the benefits of the hardware-based solution (`TranSCAMPfpga`) against the software emulation one (`TranSCAMP`). To do so, we obtain the Joules consumed by the FPGA using Xilinx's *xbutil* [43] tool, and we use *rapl-tools* [44] to obtain the Joules consumed by the CPU implementations. On the other hand, we use a transprecision FPU [36], evaluated as part of RISC-V processors and suitable for FPGAs and ASICs, to estimate the potential benefits of a transprecision FPU with respect to a double-precision FPU, which are based on the operation breakdown statistics and the energy per operation numbers given in [36].

We use real-data time series and the Top-K Accuracy metric of Definition 5 to evaluate TraTSA. Table 5 summarizes the parameters of the time series we use for the experiments. *Song* corresponds to the song *London Bridge is Falling Down* [45] converted into Mel-frequency Cepstral Coefficients, which are commonly used in speech recognition [46]. *ECG_short* and *ECG* are extracted from an electrocardiogram signal from the European ST-T Database [47]. We select the 180 000 and 1 800 000 first samples of the V4 electrode from ECG 0103, respectively. *Power_short* and *Power* are two time series extracted from fridge-freezer power consumption numbers collected over a whole year in a set of UK households [45]. *Seismology_short* and *Seismology* are two time series of seismic data collected by a seismograph in a geologically active region of the Long Valley Caldera, California [45]. We analyze the first 180 000 and 1 727 990 samples of the original series, respectively. *Human Activity* comprises a time series with information of the optical flow of an actor performing activities, from picking up an object to talking on a mobile phone [45]. *Penguin Behavior* is a time series of penguin magnetometer telemetry [48]. *Speech* is an speech recorded time series where the first author of the paper reads a fragment (2 mins) of *El Quijote* book sampled at 16 KHz. *IMU* is a time series extracted from the calibration of an Inertial-Measurement-Unit used in robotics [49]. Finally, *EPG* is insect electrical penetration graph data from [50].

5.2. Results

5.2.1. Short time series accuracy

We use the Top-K Accuracy metric of Definition 5 to evaluate our proposals. The value of K will depend on the number of significant events of a given time series, which will be eventually determined by a domain expert.

One way of setting K is defining a profile threshold for both motifs and discords. Figs. 10 and 11 present the Top-100 motif/discord accuracy with respect to double for a wide range of configurations of exponent and mantissa in `TranSCRIMP`, respectively. In contrast,

[2] Notice that both SCRIMP and SCAMP perform profile calculations (P_i) based on a previous result along the diagonals (dot product and covariance, respectively). In such scenarios, it is possible that accumulated errors provide mismatching results for large time series. However, as production implementations of SCRIMP and SCAMP are based on a tiled approach where the dot product or covariance is calculated from scratch for each 128k-512k elements of the diagonal, our evaluation and conclusions are also valid for time series of larger sizes. This is the main reason we include a recalculation factor in `TranSCAMPfpga`.

Table 5
Time series dataset parameterization.

Time series	n	m	Max	Min	Scale
Song	20 234	200	6.69	−56.48	1.00
ECG_short	180 000	500	2.60	0.32	1.00
Power_short	180 000	1 325	14.0	0.00	0.10
Seismology_short	180 000	50	6.96	−1.86	0.01
Human activity	7 997	120	2.51	−1.90	1.00
Penguin behavior	109 842	800	0.52	−0.21	1.00
Speech	1 933 944	16 384	0.98	−1.00	1.00
ECG	1 800 000	512	3.39	−1.64	1.00
Power	1 754 985	1 536	14.00	0.00	0.10
Seismology	1 727 990	64	23.29	−23.34	0.10
IMU	1 756 230	256	1.65	−2.87	1.00
EPG	2 000 000	16 384	72.26	−61.30	10.00

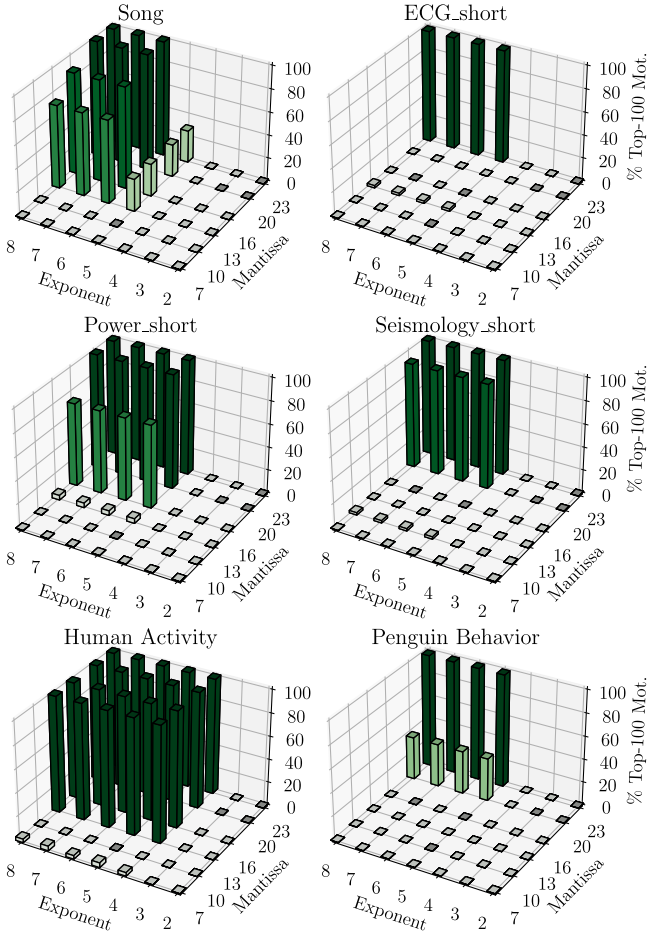

Fig. 10. TranSCRIMP Top-100 motif accuracy with respect to double.

Figs. 12 and 13 present the same metrics in TranSCAMP, respectively. In most cases, single precision (8,23) provides 100% accuracy with respect to double precision. As can be noticed, most of the plots follow a square-like shape where the accuracy decreases dramatically after a given combination of exponent and mantissa. This may occur whether one or both of the following scenarios appear: (i) the range of the exponent has been exceeded; (ii) the precision provided by the mantissa is not enough for the calculations.

Comparing both algorithms, we can observe that SCAMP is more robust and presents a better numeric stability than SCRIMP for all the datasets. This fact can be noticed for *Penguin Behavior*, where a slight decrease in the length of the mantissa makes SCRIMP fail in detecting events, while SCAMP provides more margin in this reduction. We find an outlier scenario for the time series *Power_short* in Fig. 12, where

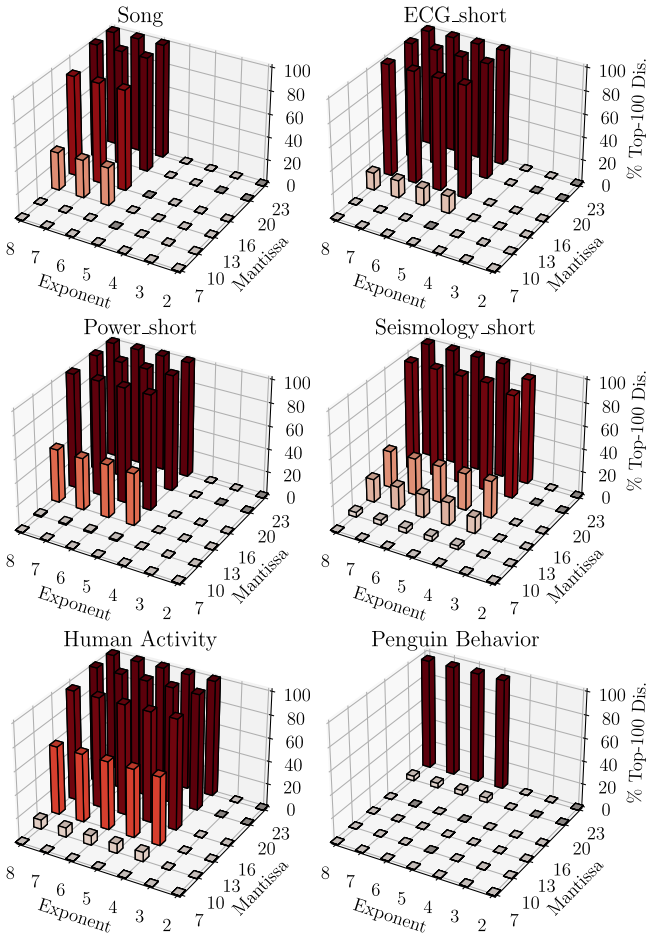

Fig. 11. TranSCRIMP Top-100 discord accuracy with respect to double.

accuracy seems to decrease when the number of mantissa bits increases from 20 to 23. The reason behind it is that, for this time series, there are several motifs with exactly the same profile value. Because of that, our sorting algorithm induces some order differences when comparing the transprecision version with respect to the reference one. However, from the practical point of view, the accuracy of the mantissa 23 is as good as the mantissa 20 since all motifs are present in both of them. We conclude that SCAMP is a better candidate for transprecision computing since it provides similar accuracy with lower bit count requirements for the exponent and mantissa.

We also evaluate how the profile is affected when reducing the exponent/mantissa bit count using TranSCAMP. Fig. 14 presents the transprecision profile for the *Song* dataset and several exponent/mantissa combinations (i.e., (7,13), (7,10), (5,13), (5,7) in the figures) with respect to a double-precision profile. We observe that for combinations of (5,13), (7,13) and (7,10) the profile is well-preserved (i.e., both curves overlap). We need to reduce the FP bit count to a lower value (e.g., (5,7)) to find mismatching results. We notice that, while in this (5,7) scenario the profile is somehow conserved with an offset, the profile index is not providing the exact matches with respect to the reference (double) solution. This fact is explained because there are potentially many values very close to each other along the profile, which can lead to high chances of profile index interchanging even if profile values slightly differ from the double-precision reference.

Fig. 15 shows the transprecision profile with respect to double for the *ECG_short* dataset. We observe that in this case the value of the mantissa plays a crucial role, since reducing it to a value lower than 13 bits leads to mismatching results (i.e., the transprecision curve is

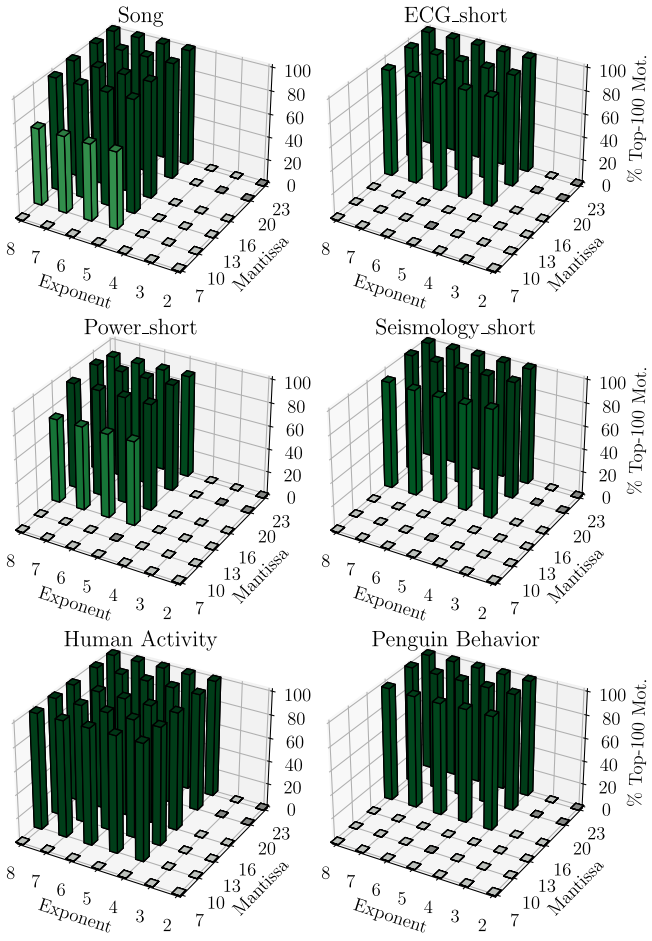

Fig. 12. TranSCAMP Top-100 motif accuracy with respect to double.

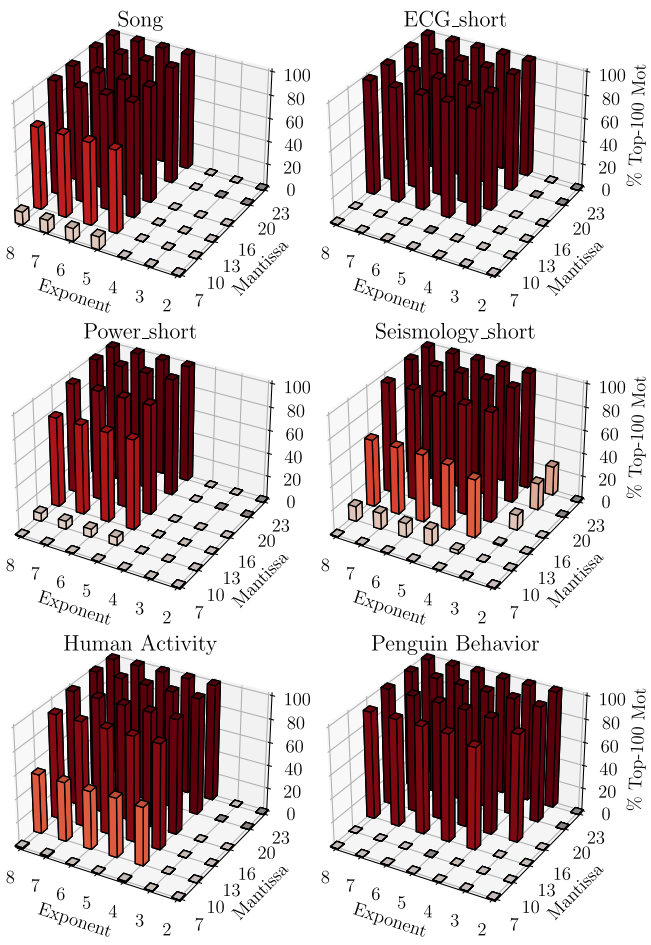

Fig. 13. TranSCAMP Top-100 discord accuracy with respect to double.

not shown at all since it contains NaN values). This fact is explained because the *ECG_short* dataset is a very regular and monotonic time series, where profile values are smaller and mantissa becomes more relevant than exponent.

Fig. 16 shows the transprecision profile with respect to double for the *Power_short* dataset. We notice that the profile is well conserved for the (7,13), (7,10) and (5,13) exponent and mantissa combinations. However, the transprecision profile diverges from the oracle solution using lower bit counts, as shown in the (5,7) scenario. This dataset also benefits from larger mantissas since it follows a regular pattern, this explains why the (5,13) combination provides better results (i.e., the transprecision profile is better preserved) than the (7,10) one.

Fig. 17 shows the transprecision profile with respect to double for the *Seismology_short* dataset. We observe a similar behavior to that in the *Song* dataset. While (7,13), (7,10) and (5,13) transprecision profiles are well-conserved with respect to double, lower mantissa bit count (e.g., 7 bits) leads to mismatching results and divergence.

Fig. 18 shows the transprecision profile with respect to double for the *Human Activity* dataset. We notice that this dataset presents a higher sensitivity to mantissa bit count, since even with 13 bits the profile does not match perfectly. However, as evaluated with the Top-K metric, the overall accuracy of the results is not affected in a higher degree than the other datasets, since *peaks* in the profile are distinguishable among them.

Fig. 19 shows the transprecision profile with respect to double for the *Penguin Behavior* dataset. This dataset presents a similar behavior to the *Power_short* one, as they have similar numeric ranges (i.e., exponent/mantissa of (7,13), (7,10) and (5,13) provide well-conserved profiles while (5,7) fails).

We also evaluate a mixed-precision approach for TranSCRIMP and TranSCAMP to further tune the bit counts required in different parts of those algorithms. In this set of experiments, we restrict the exponent/mantissa configurations to the ones available in FPnew due to space limitations. Table 6 shows the results using mixed-precision for TranSCRIMP and TranSCAMP. We obtain better results with TranSCAMP for most datasets, as expected. It is worth noting the case of *Penguin Behavior*, where TranSCRIMP is not able to detect any of the events while TranSCAMP finds near 100% of them with the (8/23, 5/10) configuration.

Overall, our evaluation shows that the mixed-precision configurations provide better accuracy results than using only one reduced precision configuration throughout the code. The experiments suggest that a mixed configuration of (8/23, 5/10) can be used for the majority of the applications analyzed in this work. In this sense, we can observe that SCAMP results for *Song* present 70% accuracy for a 5/10 configuration (see Figs. 12 and 13), whereas the accuracy increases to 95% using (8/23, 5/10) mixed precision (see Table 6). The rest of the series present a similar behavior with SCAMP: 0% accuracy for *ECG* with the 5/10 configuration, while up to 99% discord accuracy with the (8/23, 5/10)-mixed-precision configuration; 0% motif and 7% discord accuracy for *Power* with the 5/10 configuration, whereas 81% motif and 65% discord accuracy with mixed precision; and so on. SCRIMP presents a similar pattern as well.

We observe that the (8/23, 5/2)-mixed-precision configuration does not yield meaningful results for any of the analyzed datasets, since

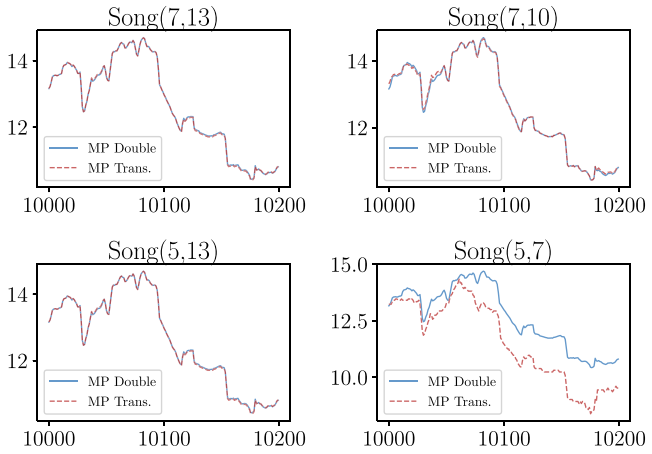

Fig. 14. TranSCAMP *Song* profile with respect to double. The horizontal axis represents the index of the datapoints within the complete time series and the vertical axis represents the amplitude of the signal.

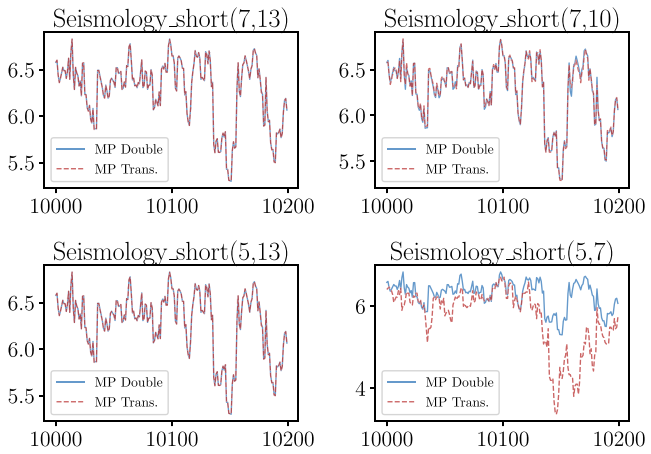

Fig. 17. TranSCAMP *Seismology_short* profile with respect to double. The horizontal axis represents the index of the datapoints within the complete time series and the vertical axis represents the amplitude of the signal.

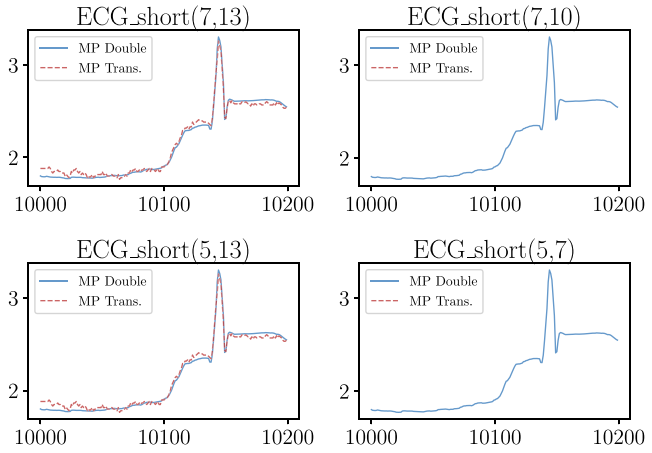

Fig. 15. TranSCAMP *ECG_short* profile with respect to double. The horizontal axis represents the index of the datapoints within the complete time series and the vertical axis represents the amplitude of the signal.

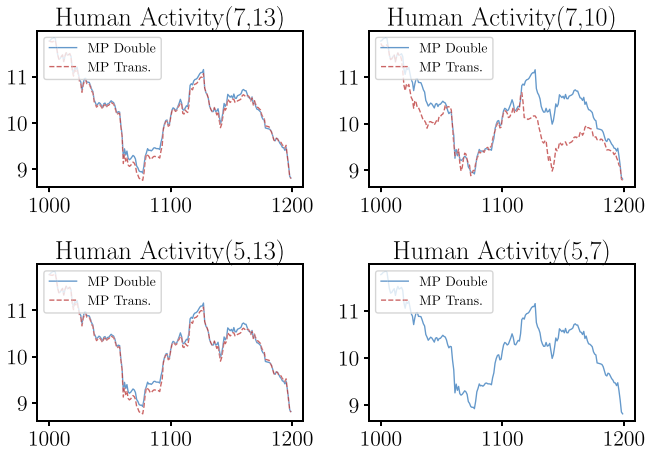

Fig. 18. TranSCAMP *Human Activity* profile with respect to double. The horizontal axis represents the index of the datapoints within the complete time series and the vertical axis represents the amplitude of the signal.

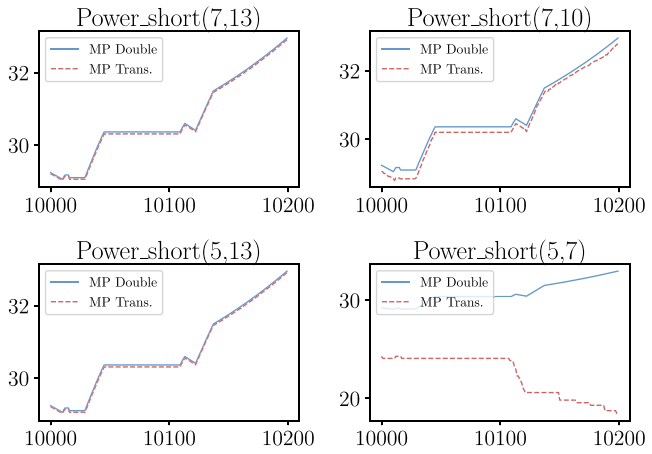

Fig. 16. TranSCAMP *Power_short* profile with respect to double. The horizontal axis represents the index of the datapoints within the complete time series and the vertical axis represents the amplitude of the signal.

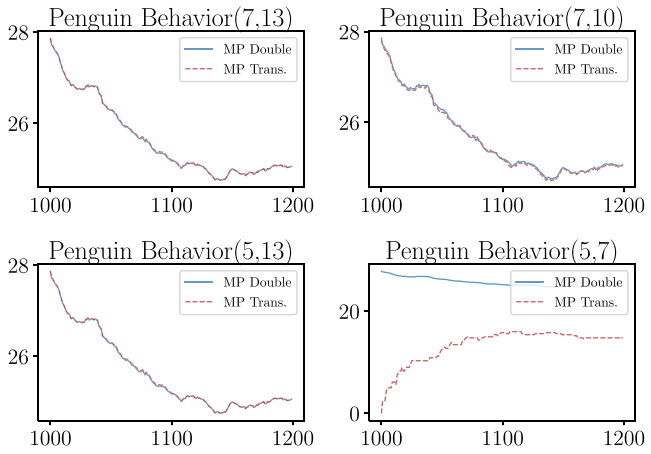

Fig. 19. TranSCAMP *Penguin Behavior* profile with respect to double. The horizontal axis represents the index of the datapoints within the complete time series and the vertical axis represents the amplitude of the signal.

the low bit count for the mantissa (only two bits) does not provide enough resolution for the correlation calculations. It can be noticed that detecting discords (anomalies) is more accurate than detecting motifs (similarities). This can be due to the fact that similarities are

Table 6
Mixed precision Top-100 accuracy results.

T	High	Low	TranSCRIMP		TranSCAMP	
	E/M	E/M	Accuracy Mot/Disc	Accu. ±10 Mot/Disc	Accuracy Mot/Disc	Accu. ±10 Mot/Disc
Song	8/23	8/7	14/9	16/31	54/86	100/97
	”	5/10	38/0	99/0	95/99	100/100
	”	5/2	0/0	0/0	1/0	1/0
ECG	8/23	8/7	0/1	0/1	0/51	0/56
	”	5/10	10/57	10/60	25/99	30/100
	”	5/2	0/0	0/0	0/0	0/0
Power	8/23	8/7	47/31	67/95	39/25	78/99
	”	5/10	68/92	96/100	81/65	100/100
	”	5/2	0/0	0/0	0/0	0/0
Seis.	8/23	8/7	3/17	55/21	0/3	0/6
	”	5/10	7/68	86/70	12/40	15/45
	”	5/2	0/0	0/0	0/0	0/0
Hum.	8/23	8/7	72/24	80/63	91/84	99/92
	”	5/10	100/85	100/97	100/98	100/99
	”	5/2	0/3	0/4	0/0	0/1
Peng.	8/23	8/7	0/0	0/0	15/89	85/98
	”	5/10	0/0	0/0	81/99	100/99
	”	5/2	0/0	0/0	0/0	0/0

low values of the profile which require more precision than discords, that typically are higher ones. This fact takes more relevance in very monotonic time series, where most of the subsequences are similar to each other (e.g., the beats in an electrocardiogram – *ECG*).

The number of detected events and its significance must be eventually determined by a domain expert. Thus, we can think that presenting a time series subsequence to an expert, in its context, and moved slightly to the left/right might end up with the expert coming to the same conclusion as if the subsequence did not move. For that reason we introduce the concept of Accuracy ±10 (see Table 6), which is the accuracy calculated for the motif/discord indices ranging in a ±10 interval. Using this metric the accuracy peaks 100% for most of the datasets.

Mixed precision configurations are aimed at balancing the trade-off between accuracy and performance (time/energy) of the matrix profile algorithms. We can reach high peaks of accuracy with the (8/23, 5/10) configuration while reducing time and energy consumption as described in Section 5.2.4.

5.2.2. Large time series accuracy

We evaluate the accuracy of the larger time series using TranSCAMPfpga. To this end, we run the larger time series (between 1M and 2M elements) using the Alveo U50 FPGA, and evaluate the results using the Top-1000 accuracy metric (i.e., increasing K proportionally to time series length). We calculate time series statistics using double precision in the host side, which increases accuracy while it does not significantly affect the total execution time (statistics only take <1 s using 1 thread in a Xeon Gold for 2M data points). This fact is particularly important in large time series, since statistics are calculated in an accumulative manner and single precision is not enough to avoid errors.

From the algorithmic point of view, TranSCAMPfpga differs from TranSCAMP in that (i) TranSCAMPfpga includes the possibility to recalculate the covariance value from scratch after certain number of elements of a diagonal, and (ii) TranSCAMP allows us to define mixed-precision configurations. The first feature resets the accumulated error due to calculation reuse across the diagonal, which increases accuracy in most datasets and allows to potentially evaluate time series of arbitrary length. We leave the mixed-precision evaluation of TranSCAMPfpga for future work.

Figs. 20 and 21 show the accuracy results obtained for motifs and discords, respectively. We observe that TranSCAMPfpga follows a

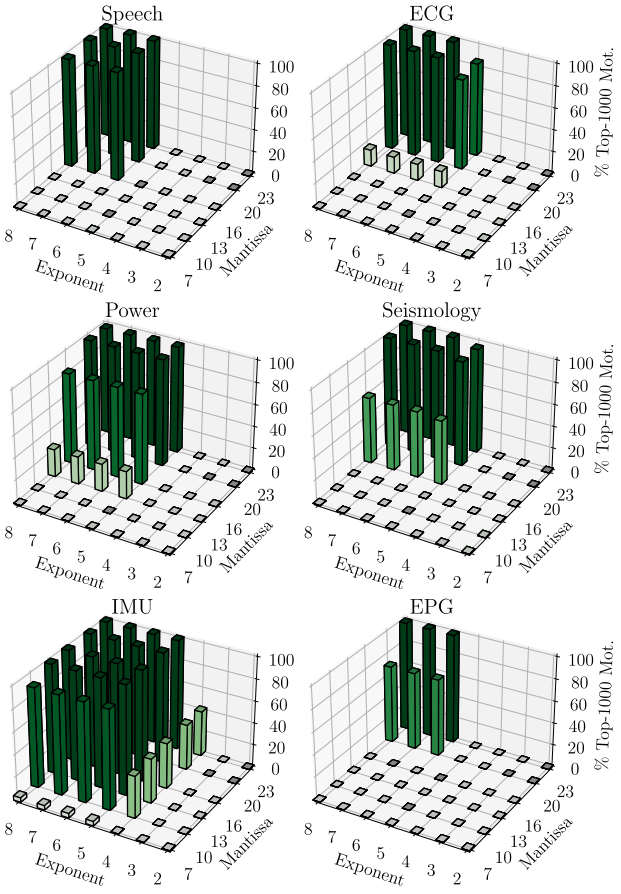

Fig. 20. TranSCAMPfpga Top-1000 motif accuracy results with respect to double precision using a recalculation factor of 64k elements.

similar behavior than the results obtained for its CPU counterpart implementation (TranSCAMP in Figs. 12 and 13). We make several observations here. First, analyzing the larger time series, we observe that single precision still provides almost 100% accuracy for all datasets when helped by covariance recalculation. Moreover, there is still margin to reduce the exponent and mantissa bit count below single precision depending on the application. Second, we find that the top part of the cubes are smaller in those series where the window size is larger (e.g., *Speech* or *EPG*). This fact is explained because larger window sizes have to deal with larger numbers, which may be out of the range of representation during calculations. And third, those datasets that present a randomness component (e.g., IMU) benefit from lower exponent and mantissa combinations, since the average distance is higher and there is no need for high precision to distinguish them.

We also evaluate the effect of changing the recalculation on the accuracy. Fig. 22 shows the % of Top-1000 accuracy for the analyzed datasets with different exponent and mantissa combinations. We analyze *recalculation factors* of every 64k and 256k elements of the diagonal against turning this feature off. We observe that 64k elements is the sweet-spot since it provides even better results than lower ones (e.g., 16k) without having great impact on performance. Lower recalculation factors imply significant impact on performance as dot product has to be calculated more frequently. In contrast, turning this feature off for series of ≈2M elements leads to mismatching results. We note that a recalculation factor of 16k for the *Seismology* time series leads to a counterintuitive result. The explanation behind it relies on the fact that the seismology time series is a very periodic and regular time series without significant complexity (and, as a consequence, many motifs and discords are highly similar). This scenario, previously observed in [51],

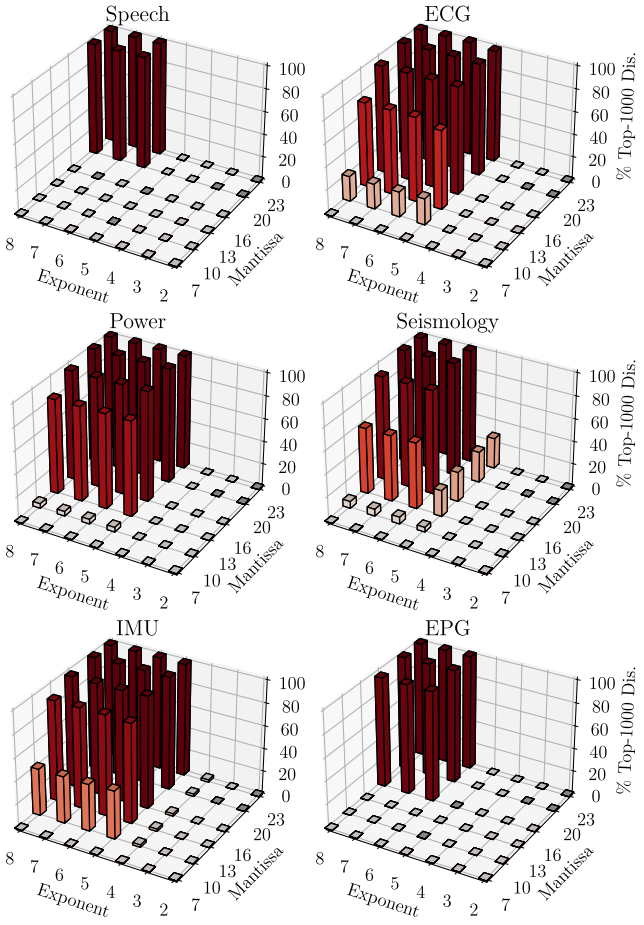

Fig. 21. `TranSCAMPfpga` Top-1000 discord accuracy results with respect to double precision using a recalculation factor of 64k elements.

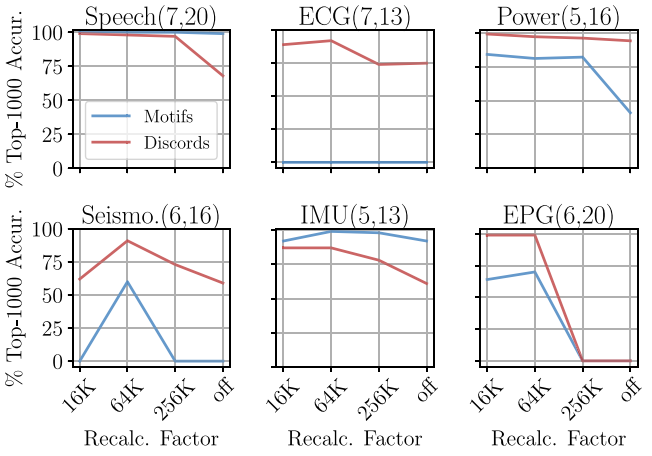

Fig. 22. `TranSCAMPfpga` Top-1000 accuracy results with respect to double precision when varying the recalculation factor.

induces a random component when generating the Top-K comparison that takes special relevance when using the lower recalculation factor for this type of time series.

We show an example of modifying the recalculation factor in Fig. 23, where we present the matrix profile of the *EPG* dataset with respect to the double-precision one. Notice that while recalculation factors of 16k and 64k provide accurate results (i.e., the curves overlap),

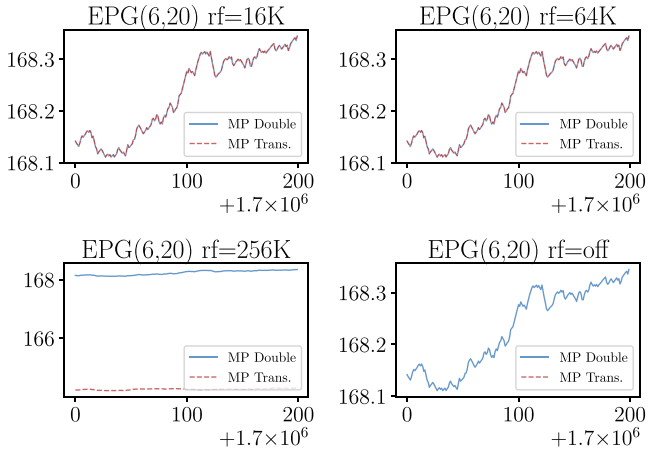

Fig. 23. EPG matrix profile when varying the recalculation factor using `TranSCAMPfpga` for a given mantissa and exponent combination.

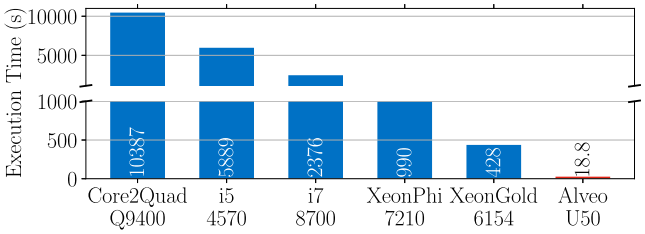

Fig. 24. Execution time for different platforms when computing *Seismology_short*, using a window size of 512 elements (exp. 7 man. 10).

a factor of 256k makes the profile always be close to 0 while turning this feature off leads to a NaN scenario.

5.2.3. TraTSA performance

We evaluate the performance of TraTSA when running in different computing platforms. We focus our attention on comparing `TranSCAMP` versus `TranSCAMPfpga`, since according to sections 5.2.1 and 5.2.2, SCAMP provides more accurate results than SCRIMP for a wider range of datasets.

Fig. 24 presents the execution times obtained when computing the *Seismology_short* series using a window of 512 elements and an exponent/mantissa combination of 7/10. Our FPGA execution times also include the memory transfers from and to the FPGA. As can be noticed, `TranSCAMPfpga` in an Alveo U50 integrated in the Xeon Gold 6154 platform outperforms `TranSCAMP` in commodity servers by 22.75× when using a 72-core Xeon Gold, and by 52.65× when using a 64-core Intel Xeon Phi KNL. Compared to desktop computers, `TranSCAMPfpga` outperforms an Intel i7-8700 by 126× and by 313× an Intel i5-4570.

5.2.4. Energy savings

Emulated vs Hardware Transprecision. First, we show an energy consumption comparison in Table 7 for the platforms in Fig. 24. As expected, the FPGA solution not only provides benefits in terms of performance but also reduced energy consumption. According to our measurements, the Alveo U50 consumes 0.47 kJ when running `TranSCAMPfpga`, taking into account the memory transfers and idle time of the host CPU. This is way less than the CPU evaluated platforms, where, for example, the most efficient one, which is the Xeon Gold server, takes 82 kJ (174× more than the FPGA). This fact demonstrates that hardware-based solutions are way more efficient than the emulated ones.

Table 7
Energy consumption for different platforms when computing *Seismology_short*, using a window size of 512 elements (exp. 7 man. 10).

	i5	i7	XeonPhi	XeonGold	Alveo
Platform	4570	8700	7210	6154	U50
Energy	180 KJ	127 KJ	132 KJ	82 KJ	0.47 KJ

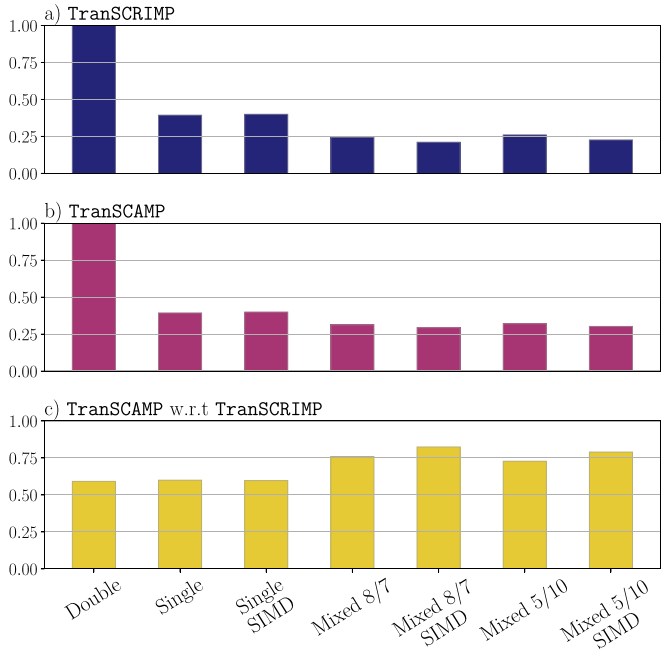

Fig. 25. Normalized FPU energy using FPNew.

Transprecision FPU. Second, we present in Fig. 25 the normalized energy consumption of (a) `TranSCRIMP` with respect to SCRIMP double, (b) `TranSCAMP` with respect to SCAMP double and (c) `TranSCAMP` with respect to `TranSCRIMP` with different precisions, respectively. As the energy is calculated with respect to the number of FPU operations performed by the algorithms, the proportion holds independently of the time series. We can observe a 60% energy reduction when using single precision instead of double for both algorithms. Furthermore, we can not only expect a reduction in time for this configuration due to an improved use of the memory hierarchy, but also because of SIMD capabilities, allowing two single precision elements computed at a time. We observe that single precision provides the same accuracy than double in the majority of cases (see Figs. 10, 11, 12, 13, 20 and 21).

It is possible to reduce the energy consumption even further using mixed precision. Our `TransSCRIMP` and `TranSCAMP` mixed precision configurations can yield up to 50% and 25% energy reduction over the single precision approach respectively. The savings for `TranSCRIMP` are more pronounced since there are more operations computed in low precision. The dot product is computed in high precision but the distance in Eq. (1) is calculated in low precision. However, the distance in `TranSCAMP` is given by the Pearson correlation coefficient in Eq. (8) which entails fewer operations. We confirm this fact when comparing `TranSCAMP` with respect to `TranSCRIMP` in plot (c) of Fig. 25, where `TranSCAMP` reduces the energy consumption between 18% and 40% for the same given precision. There are also a significant number of comparisons computed in low precision in both algorithms, although the energy cost of this operation is not as high as that of multiplications or sums, so the savings are not so high either. In contrast, the SIMD support of the FPU yields roughly the same energy numbers but opens the opportunity to improve the performance even more (up to 4 operations at a time).

6. Related work

Time series analysis. Multiple techniques for time series motif and discord discovery can be found in the literature: [10,52–57], including probabilistic solutions [10], spatio-temporal models [12], indexing [58], symbolic representation [59] or Euclidean distances [19]. A survey of time series motif discovery can be found in [16]. Regarding matrix profile [19], we find multiple efforts to increase time series analysis efficiency and performance based on commodity CPU architectures, high-performance GPUs, heterogeneous systems and even Processing-Near-Memory [60] in [22,40,61–63].

Transprecision computing. We can find the project of this paradigm in [23], a CPU transprecision library for emulation in [29], FPGA transprecision libraries in [30,64] and also hardware implementations [24,36,65–67].

Time series analysis using transprecision computing. The authors of matrix profile provide a precision evaluation in [22]. They explain how constant regions are a source of numerical instability due to zero standard deviation. Additionally, they provide a comparison between STOMP [62] and SCAMP, reporting maximum absolute error for each execution. However, this work (i) does not compare SCRIMP vs SCAMP, although they compare against GPU-STOMPopt which is based on the same diagonal approach as SCRIMP but without the anytime property; (ii) only considers standard IEEE floating-point representations, and (iii) does not discuss whether motifs and discords are conserved in a measurable manner.

To the best of our knowledge, there are no previous transprecision time series analysis works, and particularly for matrix profile. In this paper, we extend the work in [68].

7. Conclusions and future work

This work studies how time series analysis benefits from a transprecision approach, and introduces TraTSA, a framework that allows defining the exact needed precision according to the requirements of the specific application. The proposed `TranSCRIMP`, `TranSCAMP` and `TranSCAMPfpga` implementations will help the community to design energy-efficient time series analysis solutions based on transprecision RISC-V processors, FPGAs or ASICs while minimizing area and power requirements. Our FPGA-based solution is 22.75× faster than a 72-core Xeon server thanks to the hardware transprecision support and the use of optimization techniques. Additionally, we study how matrix profile algorithms can benefit from an already presented transprecision FPU. In this sense, our analysis reveals that, for a variety of applications, the energy consumption of the matrix profile algorithms is reduced up to 3.3× compared with double precision while obtaining accurate results.

Future work comprises the evaluation of the transprecision analysis of time series in complete implementations of RISC-V processors and ASIC devices, analyzing how both performance and energy are affected by this approach in a complete system environment (e.g., evaluating the benefits in terms of data movement using the reduced datatypes).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work has been supported by the Government of Spain under project PID2019-105396RB-I00, and Junta de Andalucia under projects P18-FR-3433 and UMA18-FEDERJA-197. Funding for open access charge: Universidad de Málaga / CBUA.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jocs.2022.101784.

References

[1] K. Bhaskaran, A. Gasparrini, S. Hajat, L. Smeeth, B. Armstrong, Time series regression studies in environmental epidemiology, Int. J. Epidemiol. 42 (4) (2013) 1187–1195.

[2] Z.-G. Yu, V. Anh, Time series model based on global structure of complete genome, Chaos Solitons Fractals 12 (10) (2001) 1827–1834.

[3] M. Nerlove, D.M. Grether, J.L. Carvalho, Analysis of Economic Time Series: A Synthesis, Academic Press, 2014.

[4] N. Desai, K. Dhameliya, V. Desai, Feature extraction and classification techniques for speech recognition: a review, Int. J. Emerg. Technol. Adv. Eng. 3 (12) (2013) 367–371.

[5] L. Li, X. Su, Y. Zhang, Y. Lin, Z. Li, Trend modeling for traffic time series analysis: An integrated study, IEEE Trans. Intell. Transp. Syst. 16 (6) (2015) 3430–3439.

[6] S. Firth, T. Kane, REFIT: Smart homes and energy demand reduction, 2021, www.refitsmarthomes.org/index.php/data.

[7] C.E. Yoon, O. O'Reilly, K.J. Bergen, G.C. Beroza, Earthquake detection through computationally efficient similarity search, Sci. Adv. 1 (11) (2015).

[8] C.-K. Peng, S. Havlin, H.E. Stanley, A.L. Goldberger, Quantification of scaling exponents and crossover phenomena in nonstationary heartbeat time series, Chaos 5 (1995) 82.

[9] R.H. Shumway, D.S. Stoffer, Time Series Analysis and Its Applications, fourth ed., Springer-Verlag, 2017.

[10] B. Chiu, E. Keogh, S. Lonardi, Probabilistic discovery of time series motifs, in: 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2003, pp. 493–498.

[11] E. Keogh, J. Lin, S.-H. Lee, H. Van Herle, Finding the most unusual time series subsequence: algorithms and applications, Knowl. Inf. Syst. 11 (2007) 1–27.

[12] A. McGovern, D.H. Rosendahl, R.A. Brown, K.K. Droegemeier, Identifying predictive multi-dimensional time series motifs: an application to severe weather prediction, Data Min. Knowl. Discov. 22 (2011) 232–258.

[13] C. Cassisi, M. Aliotta, A. Cannata, P. Montalto, D. Patanè, A. Pulvirenti, L. Spampinato, Motif discovery on seismic amplitude time series: the case study of mt etna 2011 eruptive activity, Pure Appl. Geophys. 170 (2013) 529–545.

[14] B. Szigeti, A. Deogade, B. Webb, Searching for motifs in the behaviour of larval drosophila melanogaster and caenorhabditis elegans reveals continuity between behavioural states, J. R. Soc. Interface 12 (2015).

[15] P. Garrard, V. Nemes, D. Nikolic, A. Barney, Motif discovery in speech: application to monitoring alzheimer's disease, Curr. Alzheimer Res. 14 (2017) 951–959.

[16] S. Torkamani, V. Lohweg, Survey on time series motif discovery, Data Min. Knowl. Discov. 7 (2017) e1199.

[17] E. Cartwright, M. Crane, H.J. Ruskin, Financial time series: motif discovery and analysis using VALMOD, in: International Conference on Computational Science (ICCS), 2019, pp. 771–778.

[18] S. Kanarachos, S.-R.G. Christopoulos, A. Chroneos, M.E. Fitzpatrick, Detecting anomalies in time series data via a deep learning algorithm combining wavelets, neural networks and Hilbert transforms, Expert Syst. Appl. 85 (2017) 292–304.

[19] C.-C.M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H.A. Dau, D.F. Silva, A. Mueen, E. Keogh, Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets, in: IEEE 16th International Conference on Data Mining (ICDM), 2016, pp. 1317–1322.

[20] R.J. Bayardo, Y. Ma, R. Srikant, Scaling up all pairs similarity search, in: 16th International Conference on World Wide Web (WWW), 2007, pp. 131–140.

[21] Y. Zhu, C.-C.M. Yeh, Z. Zimmerman, K. Kamgar, E. Keogh, Matrix profile XI: SCRIMP++: time series motif discovery at interactive speeds, in: 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2018.

[22] Z. Zimmerman, K. Kamgar, N.S. Senobari, B. Crites, G. Funning, P. Brisk, E. Keogh, Matrix profile XIV: scaling time series motif discovery with GPUs to break a quintillion pairwise comparisons a day and beyond, in: ACM Symposium on Cloud Computing (SoCC), 2019, pp. 74–86.

[23] A.C.I. Malossi, M. Schaffner, A. Molnos, L. Gammaitoni, G. Tagliavini, A. Emerson, A. Tomás, D.S. Nikolopoulos, E. Flamand, N. Wehn, The transprecision computing paradigm: concept, design, and applications, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018.

[24] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, L. Benin, A transprecision floating-point platform for ultra-low power computing, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018.

[25] Fpnew source code, 2021, https://github.com/pulp-platform/fpnew.

[26] R.M. Allen, Q. Kong, R. Martin-Short, The MyShake platform: a global vision for earthquake early warning, Pure Appl. Geophys. 177 (2020) 1699–1712.

[27] K.H.C. Li, F.A. White, T. Tipoe, T. Liu, M.C. Wong, A. Jesuthasan, A. Baranchuk, G. Tse, B.P. Yan, The current state of mobile phone apps for monitoring heart rate, heart rate variability, and atrial fibrillation: narrative review, JMIR Mhealth Uhealth 7 (2019) e11606.

[28] Tratsa's github repository, 2022, https://github.com/ivanfv/tratsa.

[29] G. Tagliavini, A. Marongiu, L. Benini, FlexFloat: a software library for transprecision computing, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (2020) 145–156.

[30] R. DiCecco, L. Sun, P. Chow, FPGA-based training of convolutional neural networks with a reduced precision floating-point library, in: International Conference on Field Programmable Technology (ICFPT), 2017.

[31] E. Vogel, G. Saravia, D. Pastén, V. Muñoz, Time-series analysis of earthquake sequences by means of information recognizer, Tectonophysics 712–713 (2017) 723–728.

[32] G.B. Moody, R.G. Mark, The impact of the MIT-bih arrhythmia database, IEEE Eng. Med. Biol. Mag. 20 (2001) 45–50.

[33] M. Jung, D.M. Mathew, C. Weis, N. Wehn, Approximate computing with partially unreliable dynamic random access memory—Approximate DRAM, in: 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC), IEEE, 2016, pp. 1–4.

[34] A. Buttari, J. Dongarra, J. Kurzak, P. Luszczek, S. Tomov, Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy, ACM Trans. Math. Software 34 (2008).

[35] X.S. Li, J.W. Demmel, D.H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S.Y. Kang, A. Kapur, M.C. Martin, B.J. Thompson, T. Tung, D.J. Yoo, Design, implementation and testing of extended and mixed precision BLAS, ACM Trans. Math. Software 28 (2002) 152–205.

[36] S. Mach, F. Schuiki, F. Zaruba, L. Benini, A 0.80 pj/flop, 1.24 tflop/sw 8-to-64 bit transprecision floating-point unit for a 64 bit RISC-v processor in 22nm FD-SOI, in: IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SOC), 2019.

[37] J. Rose, A. El Gamal, A. Sangiovanni-Vincentelli, Architecture of field-programmable gate arrays, Proc. IEEE 81 (7) (1993) 1013–1029.

[38] A. Kurth, P. Vogel, A. Capotondi, A. Marongiu, L. Benini, HERO: Heterogeneous embedded research platform for exploring RISC-v manycore accelerators on FPGA, 2017, arXiv:1712.06497 [cs.AR].

[39] D.U. Lee, K.W. Kim, K.W. Kim, H. Kim, J.Y. Kim, Y.J. Park, J.H. Kim, D.S. Kim, H.B. Park, J.W. Shin, J.H. Cho, K.H. Kwon, M.J. Kim, J. Lee, K.W. Park, B. Chung, S. Hong, 25.2 A 1.2v 8gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV, in: IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014, pp. 432–433.

[40] I. Fernandez, A. Villegas, E. Gutierrez, O. Plata, Accelerating time series motif discovery in the intel xeon phi KNL processor, J. Supercomput. 75 (2019) 7053–7075.

[41] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, K. Kim, HBM (high bandwidth memory) DRAM technology and architecture, in: IEEE International Memory Workshop (IMW), 2017.

[42] J. Jeffers, J. Reinders, A. Sodani, Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition, Morgan Kaufmann, 2016.

[43] X. Corporation, Xilinx board utility (xbutil), 2022, https://xilinx.github.io/XRT/master/html/xbutil.html.

[44] K.N. Khan, M. Hirki, T. Niemi, J.K. Nurminen, Z. Ou, Rapl in action: Experiences in using RAPL for power measurements, in: TOMPECS, 2018.

[45] C.M. Yeh, H.V. Herle, E. Keogh, Matrix profile III: the matrix profile allows visualization of salient subsequences in massive time series, in: IEEE 16th International Conference on Data Mining (ICDM), 2016.

[46] A.V. Oppenheim, R.W. Schafer, From frequency to quefrency: a history of the cepstrum, IEEE Signal Process. Mag. 21 (2004) 95–106.

[47] A. Taddei, G. Distante, M. Emdin, P. Pisani, G.B. Moody, C. Zeelenberg, C. Marchesi, The European ST-t database: standard for evaluating systems for the analysis of ST-t changes in ambulatory electrocardiography, Eur. Heart J. 13 (1992) 1164–1172.

[48] Penguin data, 2021, https://www.cs.ucr.edu/~eamonn/MatrixProfile.html.

[49] D.Z. niga Nol, A. Jaenal, R. Gomez-Ojeda, J. Gonzalez-Jimenez, The UMA-VI dataset: visual–Inertial odometry in low-textured and dynamic illumination environments, Int. J. Robot. Res. 39 (9) (2020) 1052–1060.

[50] S. Alaee, K. Kamgar, E. Keogh, Matrix profile XXII: exact discovery of time series motifs under DTW, in: IEEE 16th International Conference on Data Mining (ICDM), 2020.

[51] H.A. Dau, E. 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, 2017, pp. 125–134.

[52] P. Patel, E. Keogh, J. Lin, S. Lonardi, Mining motifs in massive time series databases, in: IEEE International Conference on Data Mining (ICDM), 2002.

[53] Y. Tanaka, K. Iwamoto, K. Uehara, Discovery of time-series motif from multi-dimensional data based on MDL principle, Mach. Learn. 58 (2005) 269–300.

[54] D. Yankov, E. Keogh, J. Medina, B. Chiu, V. Zordan, Detecting time series motifs under uniform scaling, in: 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Miningi (KDD), 2007, pp. 844–853.

[55] H. Tang, S.S. Liao, Discovering original motifs with different lengths from time series, Knowl.-Based Syst. 21 (2008) 666–671.
[56] P. Nunthanid, V. Niennattrakul, C.A. Ratanamahatana, Discovery of variable length time series motif, in: 8th Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI) Association of Thailand - Conference, 2011.
[57] A. Balasubramanian, J. Wang, B. Prabhakaran, Discovering multidimensional motifs in physiological signals for personalized healthcare, J. Sel. Top. Signal Process. 10 (2016) 832–841.
[58] B.K. Yi, C. Faloutsos, Fast time sequence indexing for arbitrary $\mathcal{L}_p$ norms, in: 26th International Conference on Very Large Data Bases (VLDB), 2000, pp. 385–394.
[59] J. Lin, E. Keogh, L. Wei, S. Lonardi, Experiencing SAX: a novel symbolic representation of time series, Data Min. Knowl. Discov. 15 (2007) 107–144.
[60] O. Mutlu, S. Ghose, J. Gómez-Luna, R. Ausavarungnirun, A modern primer on processing in memory, 2020, arXiv:2012.03112 [cs.AR].
[61] J.C. Romero, A. Vilches, A. Rodríguez, A. Navarro, R. Asenjo, Scrimpco: scalable matrix profile on commodity heterogeneous processors, J. Supercomput. 76 (2020) 9189–9210.
[62] Y. Zhu, Z. Zimmerman, N.S. Senobari, C.-C.M. Yeh, G. Funning, A. Mueen, P. Brisk, E. Keogh, Matrix profile II: exploiting a novel algorithm and GPUs to break the one hundred million barrier for time series motifs and joins, in: IEEE 16th International Conference on Data Mining (ICDM), 2016.
[63] I. Fernandez, R. Quislant, E. Gutiérrez, O. Plata, C. Giannoula, M. Alser, J. Gómez-Luna, O. Mutlu, NATSA: A near-data processing accelerator for time series analysis, in: 38th IEEE International Conference on Computer Design (ICCD), 2020.
[64] D.B. Thomas, Templatised soft floating-point for high-level synthesis, in: IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2019.
[65] G. Stazi, F. Silvestri, A. Mastrandrea, M. Olivieri, F. Menichelli, Synthesis time reconfigurable floating point unit for transprecision computing, in: International Conference on Applications in Electronics Pervading Industry, Environment and Society (ApplePies), 2018.
[66] A. Borghesi, G. Tagliavini, M. Lombardi, L. Benini, M. Milano, Combining learning and optimization for transprecision computing, in: 7th ACM International Conference on Computing Frontiers (CF), 2020.
[67] C. Sudarshan, J. Lappas, C. Weis, D.M. Mathew, M. Jung, N. Wehn, A lean, low power, low latency DRAM memory controller for transprecision computing, in: 20th International Conference on Embedded Computer Systems (SAMOS), 2019.
[68] I. Fernandez, R. Quislant, E. Gutierrez, O. Plata, Energy-efficient time series analysis using transprecision computing, in: IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2020.

Ivan Fernandez received his B.S. degree in Computer Engineering and his M.S. degree in Mechatronics Engineering from University of Malaga in 2017 and 2018, respectively. He is currently working toward the Ph.D. degree at the University of Malaga. His current research interests include processing in memory, near-data processing, stacked memory architectures, high-performance computing, transprecision computing and time series analysis.

Ricardo Quislant received the M.Sc. degree in computer engineering from the University of Granada, and the Ph.D. from the University of Malaga, Spain, in 2006 and 2012, respectively. Currently, he is working as an assistant professor in the Department of Computer Architecture at the University of Malaga. His main research interests include computer memory system and high-performance computing, with special regard to transactional memory.

Sonia Gonzalez-Navarro received the B.S. and M.S. degrees in Mathematics in 2000 and the Ph.D. degree in Computer Science in 2006, both from the University of Malaga, Spain. She is currently an assistant professor in the Computer Architecture Department at the University of Malaga. Her research interests include computer arithmetic, floating point number computation, high-performance architectures for data-intensive applications and near-data processing.

Eladio Gutierrez received the M.Sc. and Ph.D. degrees in Telecommunication Engineering both from the University of Malaga, Spain, in 1995 and 2001 respectively. Since 2003 he has been an associate professor in the Department of Computer Architecture of the University of Malaga. His research interests include parallel architectures and algorithms, graphics processing units and automatic parallelization.

Oscar Plata earned a M.S. in Physics in 1985 and a Ph.D. in Physics in 1989, both from the University of Santiago de Compostela, Spain. He started as Assistant Professor in the same University where he became Associated Professor in 1990. He moved to the University of Malaga in 1995, where he became Full Professor in the Computer Architecture Dept. in 2002. His research interests are related to high performance computing and parallel architectures.