Time Series Analysis
Using Transprecision Computing

Ivan Fernandez Vega
NiPS Summer School
4 September 2019
About me

- 2nd-year **PhD Student at University of Malaga** (Spain)
- Advisors: Oscar Plata and Eladio Gutierrez
- Research topic: **Acceleration of time series analysis**
- Currently at **ETH Zürich** as an academic guest in SAFARI group, supervised by Prof. Onur Mutlu
Outline

- Introduction
- Background
- Implementation
- Results
- Conclusions and Future Work
Introduction
Introduction

- **Time series analysis** has a huge interest in many fields
  - Climate
  - Seismology
  - Entomology
  - Bioinformatics
  - Traffic Prediction
  - Voice Recognition
  - Energy Conservation

![Reconstructed Temperature Graph](image)
Introduction

- **Matrix Profile (from UCR Riverside)**
  - Open source tool for **motif discovery** (anomalies, similarities, ...)
  - Implemented in several languages: C++, Python, CUDA, R, MATLAB

![Matrix Profile website](QRcode)
**Introduction**

- **Similarity** example

  ![Time series graph](image)

  **Observation**
  Similarities appear as lower values of Matrix Profile

- **Anomaly** example

  ![Time series graph](image)

  **Observation**
  Anomalies appear as higher values of Matrix Profile
Motivation

- Real data example: **electrocardiogram**

- In this case there are **two anomalies** annotated by MIT cardiologists

- Here the subsequence length was set to 150, but we still find these anomalies if we half or double that length

Anomaly: ectopic beat

Anomaly: premature ventricular contraction
Motivation

- Typical data type used for the computation is **double** precision, while the algorithm allows for **single** or mixed **precision**

- **No previous study using** lower precision or **flex float approach**

- Analysing a time series of **131,072** elements using a window size of **1,024** elements requires:

  
  2.4 Billion subtractions (-)  
  2.7 Billion multiplications (*)  
  2.9 Billion divisions (/)  
  2.8 Billion multiply-accumulations (FMA)  
  + 2.8 Billion comparisons (<)  

**13.6 Billion operations !!!**
Background
Distance Matrix

Matrix Profile: a vector of distance between each subsequence and its most similar one

- Symmetric matrix
- Main diagonal = 0
- Cells close to diagonal very small
Distance metric

- The similarity $d_{i,j}$ is based on Euclidean distances:

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

- The dot product ($Q_{i,j}$) can be calculated as follows:

$$T_i T_j = \sum_{k=0}^{m-1} t_{i+k} t_{j+k}$$

$$T_{i+1} T_{j+1} = T_i T_j - t_i t_j + t_{i+m} t_{j+m}$$

$O(1)$ time complexity
Implementation
Goal

- The goal is to provide a benchmark to explore how the accuracy of the results of SCRIMP are affected by changing the precision of the floating-point operations.

- This tool would be useful for architects when designing a custom accelerator for time series analysis.

- The implementation is open source and based on FlexFloat.
SCRRMP FF computation scheme and configuration parameters

- Precomputation of statistics
- Initialization of matrix profile
- Dot product calculation or update
- Distance calculation
- Matrix profile update
- More diagonals?

The user can configure individually the precision for each block via a config file.

Configurable precisions:
- Statistics (μ and σ)
- Dot product
- Distance
- Matrix profile
SCRIMP FF example input:

./scrimp_ff

power_demand.txt

binary
time series file

200
window size

72
# threads

0.1
scale factor

SCRIMP FF example output:

original time series

result using original implementation (64 bits)

absolute error %

result using FlexFloat implementation

FF parameters [exp, man] => distance=[7, 16]; dotprod=[7, 16]; stats=[6, 12]; profile=[5, 2]
Results
Experiments

- The benchmark has been tested using a server equipped with two Intel Xeon Gold 6154 (72 threads) and 384 GB of DDR4 memory.

- Each FlexFloat execution is compared with the original code.

- In this presentation I cover four didactical examples:
  - (1) Synthetic random time series with one anomaly
  - (2) Synthetic random time series with two (very) similar subsequences
  - (3) Real data time series with four anomalies
  - (4) Real data time series with twelve anomalies

Computing a 32,768 elements time series takes approx. 4 minutes in this sever.
Random Serie Anomaly

- **Case study #1**
  - Random time series
  - Values from 0 to 100
  - 32,768 elements
  - 50 window size length
  - One anomaly
Random Serie Anomaly - 64 Bits

Observation
Using 64-bit precision and Flex Float we obtain no error, as expected

Random Serie Anomaly - 32 Bits

Observation
Using 32-bit precision and Flex Float we still obtain no error!!

FF parameters [exp, man] => distance=[8, 23]; dotprod=[8, 23]; stats=[8, 23]; profile=[8, 23]
Random Serie Anomaly - Reduced

**Observation**
When we reduce significantly the precision we get just ~10% error.

**FF parameters** [exp, man] => distance=[6, 15]; dotprod=[6, 10]; stats=[6, 12]; profile=[6, 1]
The anomaly is very easily detectable using the Flex Float approach.
Case study #2
- Random time series
- Values from 0 to 100
- 32,768 elements
- 50 window size length
- Two (very) similar subsequences
Random Serie Similarity - 64 bits

Observation
Using 64-bit precision and Flex Float we obtain no error, as expected.

Random Series Similarity - 32 Bits

**Observation**
Using 32-bit precision and Flex Float we still obtain no error!!

FF parameters [exp, man] => distance=[8, 23]; dotprod=[8, 23]; stats=[8, 23]; profile=[8, 23]
Observation

We obtain error in the lower values, however they are still detectable.

FF parameters [exp, man] => distance=[6, 17]; dotprod=[6, 15]; stats=[6, 10]; profile=[5, 1]
The similarities are still detectable using Flex Float.
Case study #3

- Taxi demand data
- 3,600 elements
- 50 window size length
- Four anomalies
Taxi Demand Data - 64 Bits

Observation
Using 64-bit precision and Flex Float we obtain no error, as expected.

Taxi Demand Data - 32 Bits

Observation
Using 32-bit precision and Flex Float we still obtain no error!

FF parameters [exp, man] => distance=[8, 23]; dotprod=[8, 23]; stats=[8, 23]; profile=[8, 23]
We obtain error in lower values, but anomalies are still detectable.

**Observation**

FF parameters [exp, man] = distance=[7, 16]; dotprod=[7, 16]; stats=[6, 12]; profile=[5, 2]
Observation
The anomalies are still detectable using Flex Float
Case study #4
- Electric power demand data
- 30,000 elements
- 50 window size length
- Twelve anomalies
Observation
Using 64-bit precision and Flex Float we obtain no error, as expected.
Power Demand Data - 32 Bits

Observation
Using 32-bit precision and Flex Float we still obtain no error!!

FF parameters \([\text{exp, man}] = [8, 23];\ \text{dotprod} = [8, 23];\ \text{stats} = [8, 23];\ \text{profile} = [8, 23]\)
We obtain error in lower values, but anomalies are still detectable.

Observation:

**FF parameters** [exp, man] => distance=[6, 17]; dotprod=[6, 17]; stats=[6, 17]; profile=[5, 2]
The anomalies are still detectable using Flex Float.
Conclusions and Future Work
Conclusions and Future Work

- Matrix profile can be useful for **many** time series motif discovery applications

- SCRIMP FlexFloat benchmark allows the **exploration of reduced precision** computation of Matrix Profile

- Architects could design accelerators using the exact amount of precision needed for each application, **maximizing performance** and **minimizing energy consumption**

- **Future work** comprises evaluating time series analysis using a non emulated transprecision computing environment as pulp-platform
Some of the examples are taken from the Matrix Profile tutorial available at https://www.cs.ucr.edu/~eamonn/MatrixProfile.html

**SCRIMP:**
- https://sites.google.com/site/scrimpplusplus/

**FlexFloat:**
- https://github.com/oprecomp/flexfloat
Time Series Analysis
Using Transprecision Computing

Ivan Fernandez Vega
NiPS Summer School
4 September 2019
Backup Slides
Matrix Profile implementation (SCRIMP)

- Takes advantage of the dot product from the previous step performing the calculations in **diagonals** instead of columns or rows:

\[
Q_{i,j} = Q_{i-1,j-1} - t_{i-1} t_{j-1} + t_{i+m-1} t_{j+m-1}
\]

\[
\begin{array}{cccc}
D_1 & D_2 & \cdots & D_{n-m+1} \\
D_1 & d_{1,1} & d_{2,1} & \cdots & d_{n-m+1,1} \\
D_2 & d_{1,2} & d_{2,2} & \cdots & \cdots \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
D_{n-m+1} & d_{1,n-m+1} & \cdots & \cdots & D_{n-m+1,n-m+1} \\
\end{array}
\]

\[
\begin{array}{cccc}
P & \min(D_1) & \min(D_2) & \cdots & \min(D_{n-m+1}) \\
I & j \ | \ \min(D_1) = d_{1,j} & j \ | \ \min(D_2) = d_{2,j} & \cdots & j \ | \ \min(D_{n-m+1}) = d_{n-m+1,j}
\end{array}
\]
Parallelization

- SCRIMP is highly parallelizable (no calculus dependency between diagonals)

- However, elements inside diagonals need results from the previous step

- Two possible computation approaches:
  - **Random order for the diagonals**
    - Benefit: allows the possibility of obtaining partial (maybe enough accurate) results if the program is interrupted
    - Drawback: less performance if complete solution needed
  - **Sequential order for the diagonals**
    - Benefit: better performance in complete solution
    - Drawback: if the program is interrupted, only part of the time series is explored
SCRIMP FF code transformation examples

Original Code

```c
for (int w = 0; w < win; w++)
{
    lastz += tSeries[w + subseq] * tSeries[w];
}
```

```c
distance = 2 * (windowSize - (lastz - windowSize * AMean[j] * AMean[i]) / (ASigma[j] * ASigma[i]));
```

FlexFloat code

```c
for (int w = 0; w < win; w++)
{
    ff_fma(&lastz, &tSeries[w + subseq],
            &tSeries[w], &lastz);
}
```

```c
ff_mul(&sigma_prods, &ASigma[subseq], &ASigma[0]);
ff_mul(&mean_prods, &AMean[subseq], &AMean [0]);
ff_cast(&mean_cast, &mean_prods,
       (flexfloat_desc_t) {dist_exp, dist_man});
ff_cast(&sigma_cast, &sigma_prods,
       (flexfloat_desc_t){dist_exp, dist_man});
ff_mul(&distance, &mean_cast, &WindowSize);
ff_sub(&distance, &lastz_cast, &distance);
ff_div(&distance, &distance, &sigma_cast);
ff_sub(&distance, &WindowSize, &distance);
ff_mul(&distance, &distance, &constant_2);
```