MESSI SOFA - study project

Gerrit Slomma

March 2025

1 Introduction

MESSI [2] is an index for exact time series similarity search introduced by Peng et al in 2020. Similarity searches are employed during all time series analysis, be it the analysis of one single time series for itself, one time series compared to others, or the analysis of multiple time series. Similarity search is a core building block of operations such as querying, clustering, segmentation, classification, or anomaly detection. To speed up similarity search an index is formed from the time series to be analyzed, split into the two categories exact search and approximate search. SAX-empbeddings are typically employed to speed up index searches but could experience degraded performance for data having background noise or a high frequency. The SOFA-implementation [3] based on MESSI utilizes the vector registers of current x86-CPUs and employs fast Fourier transform (FFT) via the fftw-library[1].

2 Current situation

The MESSI/SOFA project at hand was created as a follow up to a bachelor thesis "Implementation of SFA in the MESSI framework" by Jakob Brand. Associated with the project is a GitHub-repository, located at [2]. The core source code is written in C, but accompanied by a python framework for analyzing results that are generated during the runtime of the MESSI application. Automake is used as the build framework for the C-code, but the build

[1] `https://www.fftw.org/index.html`

[2] `https://anonymous.4open.science/r/MESSI-SFA-C4D3/`

process is not generalized that much and needs some manual work in order to get a working application. For expamle as a dependency the fftw-library in version 3 is needed, but neither provided nor checked for existence by the configure script. Additionally needed after having a working application are benchmark data series, which are not provided by the GitHub-repository. Furthermore, the source code of the project may exhibit unoptimized, ungeneralized or unused code paths. Unoptimized code paths are considered in regard to the instruction set usage the available hardware provides.

3 Scope of this work

The scope of this work comprises the following tasks:

- Analyzation and generalization of the build process
- Check for ungeneralized code paths
- Check for unused code paths
- Check for unoptimized code paths
- Possibly implementation of incremental update algorithm
- Possibly integration of build process into existing python framework
- Report of work done and final presentation

The build-process is to be analyzed and to be generalized as much as possible to facilitate easy building of the working application from the source code with as little manual work as possible. Other work done will be as follows: The source code will be checked for ungeneralized code paths and those code paths found would be changed to a more generalized implementation. Additionally a check for unused code paths will be performed and found occurrences will be reported and cleaned from the source code. This check will utilize the benchmark data sets available from the gruenau-servers. Where possible the code or the built application will be checked for unoptimized code, found unoptimized code will be reported and changed to an optimized implementation that better utilizes the hardware used, i.e. exploiting available AVX-512-registers if this provides a performance gain. Where

unoptimized code is found in dependencies, e.g. the fftw-library, those instances are merely reported, since refactoring third party dependencies is beyond the scope of this work. The Intel Software Development Emulator (SDE)[3] is used in conjunction with the Intel Software Developer Manual [1], processor white papers as well as [4] to evaluate the code optimizations and generalizations. Test runs will be performed to evaluate the changes done against the baseline.
MESSI in the current form is optimized for bulk loading and therefore does not allow updates once the index is built. This also means the entire data set has to be processed on every load, no intermittent snapshots or indexes are saved and could be taken into account when restarting the application after it was terminated. Therefore additionally the implementation of an incremental update algorithm for the application will be evaluated and possibly implemented, if available time is permitting. The possibility of including the automatized and generalized build-process into the existing python framework will be evaluated and integrated, if time is permitting. A report on the work done is compiled and a final presentation with key points from the work done is held upon finalization of the work done. The time frame allotted comprises a semester project as defined in the study regulations for Mono Master Informatik of Humboldt-Universität zu Berlin.

References

[1] I. Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2 (2A, 2B, 2C, & 2D): Instruction Set Reference, A-Z*, 2023. `https://cdrdv2-public.intel.com/774492/325383-sdm-vol-2abcd.pdf`.

[2] B. Peng, P. Fatourou, and T. Palpanas. Messi: In-memory data series indexing. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 337–348, 2020.

[3] P. Schäfer, J. Brand, U. Leser, B. Peng, and T. Palpanas. Fast and exact similarity search in less than a blink of an eye. *CoRR*, abs/2411.17483, 2024.

[3] `https://www.intel.com/content/www/us/en/developer/articles/tool/software-development-emulator.html`

[4] `https://uops.info/table.html`