

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

Efficient Transform Algorithms for Parallel Ultra-Low-Power IoT End Nodes

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:

Mazzoni B., Benatti S., Benini L., Tagliavini G. (2021). Efficient Transform Algorithms for Parallel Ultra-Low-Power IoT End Nodes. IEEE EMBEDDED SYSTEMS LETTERS, 13(4), 210-213 [10.1109/LES.2021.3065206].

Availability:

This version is available at: <https://hdl.handle.net/11585/828855> since: 2021-07-27

Published:

DOI: <http://doi.org/10.1109/LES.2021.3065206>

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (<https://cris.unibo.it/>).
When citing, please refer to the published version.

(Article begins on next page)

This is the final peer-reviewed accepted manuscript of:

B. Mazzoni, S. Benatti, L. Benini and G. Tagliavini, "Efficient Transform Algorithms for Parallel Ultra-Low-Power IoT End Nodes," in IEEE Embedded Systems Letters, vol. 13, no. 4, pp. 210-213, Dec. 2021

The final published version is available online at
<https://dx.doi.org/10.1109/LES.2021.3065206>

Rights / License:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (<https://cris.unibo.it/>)

When citing, please refer to the published version.

Efficient Transform Algorithms for Parallel Ultra-Low-Power IoT End Nodes

Benedetta Mazzoni, Simone Benatti, Luca Benini, *Fellow, IEEE*, Giuseppe Tagliavini, *Member, IEEE*

Abstract—Modern IoT end nodes must support computational intensive workloads at a limited power-budget. Parallel ultra-low-power architectures are a promising target for this scenario, and the availability of highly optimized software libraries is crucial to exploit parallelism and reduce software development costs. This letter proposes an efficient parallel design of the widely used STFT and DWT transforms targeting ultra-low-power IoT devices. We address key performance challenges related to fine-grained synchronization and banking conflicts in shared memory. We achieve high throughput (50.95 samples/ μ s, on average), good parallel speedup (up to 6.79 \times), and high energy efficiency (up to 172.55 GOp/s/W) on a cluster of 8 RISC-V cores optimized for parallel ultra-low-power (PULP) operation.

Index Terms—STFT, DWT, IoT, parallel programming

I. INTRODUCTION AND RELATED WORK

IN recent years, ultra-low-power (ULP) parallel computing platforms based on the RISC-V instruction set architecture (ISA) have proved to be an effective solution for Internet of Things (IoT) end nodes, as an open alternative to proprietary-ISA microcontroller units (MCUs) (e.g., ARM Cortex-M4) [1] [2]. The *parallel ultra-low-power platform* (PULP) [3] is an open-source hardware project aiming to provide a RISC-V programmable architecture with the primary goal to meet the computational requirements of IoT applications within a power envelope of 10 mW. The recent embodiments of this architecture include a control core dedicated to I/O and system management, coupled with a cluster of cores sharing a tightly-coupled data memory (TCDM). The PULP approach enables operating the cluster at the energy-optimal operating voltage (i.e., near-threshold [4]) while achieving high computational throughput thanks to parallel execution [5].

The availability of efficient shared-memory parallel software libraries for fundamental algorithmic kernels is a key enabler to fully exploit ULP platforms and reduce software costs. For traditional single-core MCUs, CMSIS-DSP [6] is a hardware abstraction layer (HAL) targeting ARM Cortex-M cores, which provides a set of optimized digital signal processing (DSP) kernels. A key challenge in developing similar libraries for PULP is to achieve a good parallel speed-up, which is essential for obtaining high energy efficiency.

In this letter, we discuss the parallel design of two DSP algorithms: short-time Fourier transform (STFT) and discrete

wavelet transform (DWT). DSP applications make pervasive use of the 1-D floating-point variants of these algorithms: They enable the extraction of relevant features on time and frequency domains serving as pre-processing stages for machine learning methods. In real-life use cases, DWT is used in [7] to extract features from physiological data in a pattern recognition application that relies on an embeddable support vector machine (SVM). STFT is used in [8] for structural anomaly detection, providing the time-frequency analysis of current consumption, voltage, and vibrations of industrial equipment. Since machine learning models adopted for near-sensor processing, such as multi-layer perceptron (MLP) [9] and SVM [10], are amenable to lightweight designs executing in a few thousand cycles, optimizing the pre-processing stages based on DWT and STFT is crucial to improve performance and energy efficiency.

The main block of STFT is the fast Fourier transform (FFT) algorithm. FFTW [11] is the most widespread FFT implementation, and it is widely used in scientific computing. However, this library has a complex design, and embedded system designers do not commonly adopt it for performance reasons. In most cases, lightweight FFT libraries are not portable and do not provide optimized parallel support. For instance, Kiss FFT [12] is parallelized using OpenMP directives, but it is not optimized. The GNU scientific library (GSL) [13] provides a parametric implementation of the DWT algorithm even though it does not provide any support for code parallelization. Moreover, to the best of our knowledge, there is no DWT implementation specific to the embedded domain.

In this letter, we discuss three main contributions. First, we describe an algorithm design for FFT and DWT focused on performance optimization on ULP IoT end nodes. This goal requires a fine-grain analysis to maximize the instructions per cycle (IPC) for each processing thread. We provide specific insight into this methodology. Second, we provide an experimental assessment on an 8-core PULP cluster with 4 floating-point units (FPUs), analyzing the impact of the key design optimizations. Finally, we provide a comparison with a Cortex-M4 platform and alternative libraries (GSL and Kiss FFT).

II. METHODOLOGY AND DESIGN

A. Parallel programming methodology

Since the cluster cores are independent and execute separate instruction flows, the programming interface supports the single-program multiple-data (SPMD) paradigm. The PULP HAL [14] provides two main concepts: *core identifiers* and *barriers*. The core identifier is a fundamental mechanism to split the workload among multiple execution flows (*parallel*

B. Mazzoni, S. Benatti and L. Benini are with the Dept. of Electrical, Electronic, and Information Engineering (DEI), University of Bologna (IT). E-mail: {benedetta.mazzoni3, simone.benatti, luca.benini}@unibo.it

L. Benini is also with the Dept. of Information Technology and Electrical Engineering (D-ITET), ETH Zürich (CH). E-mail: benini@iis.ee.ethz.ch

G. Tagliavini is with the Dept. of Computer Science and Engineering (DISI), University of Bologna (IT). E-mail: giuseppe.tagliavini@unibo.it

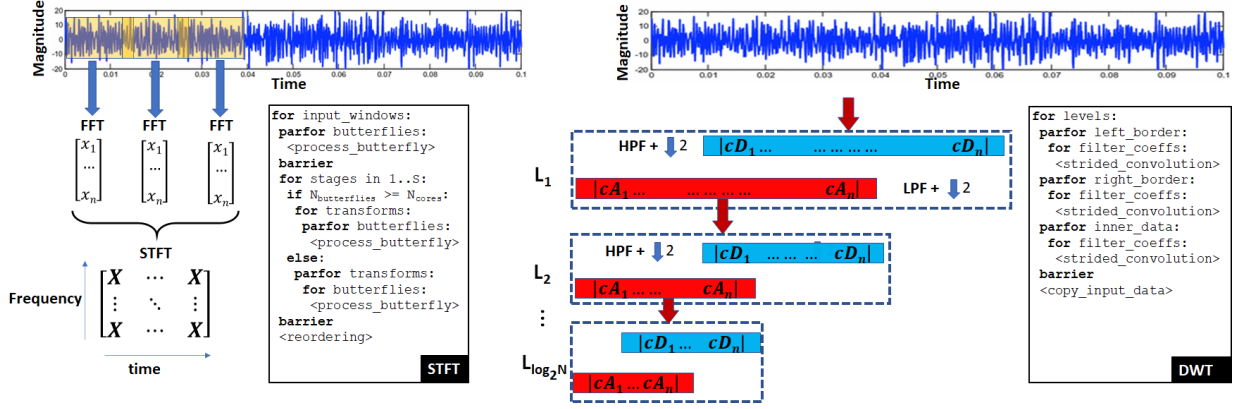


Fig. 1. Structural and working flow diagrams of STFT and DWT.

orchestration). For instance, programmers can employ loop-level parallelism using core identifiers into the control expressions (i.e., initialization, condition check, and increment). A barrier is a HAL function that stops a core until all other cores arrive at the same execution point. Barriers are synchronization points that guarantee data consistency between adjacent code regions (before and after the barrier). The *event unit* [15] is a dedicated hardware component providing low-overhead support for barriers and enabling the adoption of power-saving policies when cores are waiting. Thanks to this specialized component, the experimental evaluation in Section III reports small overheads related to synchronization points.

Fig. 1 depicts the adoption of loop-level parallelism (*parfor*) and barriers (*barrier*) in the design of the algorithms, further described in the next subsection.

B. STFT and DWT Algorithm Design

Our design of STFT is illustrated in Fig. 1. Input buffer size, number of data samples, and overlap size are configurable parameters. The most relevant kernel of the STFT is FFT calculation, based on the mixed-radix variant of the decimation-in-frequency Cooley–Tukey algorithm, a solution also used by Kiss FFT and CMSIS. This class of algorithms recursively breaks down a transform on input with size $N = r \times m$ into r smaller transforms of size m . Each recursive call is called a *stage*, and transforms of size r are generally referred as *butterflies*. Our design adopts a *mixed-2-8* variant that applies a radix-8 FFT when the size of the input is a power of eight; otherwise, it performs one or two preliminary radix-2 stages.

The first FFT stage ($N/2$ butterflies) can be equally split among the available cores. Each of the following stages includes $2^s \times m$ transform step, where s is the zero-based stage index. The butterflies inside a transform step are equally split among the cores if they are enough to guarantee workload balancing – i.e., m/r must be greater or equal to the number of available cores. Otherwise, transform steps are partitioned in disjoint sets that are distributed among the cores. This approach guarantees workload balancing in all cases, and it also minimizes the overhead of the parallel orchestration since workload distribution is always associated with a single loop. Each stage requires a single barrier at the end to guarantee data consistency for the next one.

Decimation-in-frequency algorithms require output reordering as a final stage. Index remapping is provided by a pre-computed look-up table so that this task can be equally split among the cores. However, access to the look-up table and subsequent swap operations are highly memory-bound and cause TCDM stalls. To hide this latency, we applied *loop unrolling* to the reordering outer loop.

DWT is a time-frequency analysis technique relying on a pair of recursive convolutions, which decompose the original signals extracting its low and high frequency contents, referred to the time domain [16]. As depicted in Fig. 1, for a given input signal of length N , DWT applies the two convolutions followed by dyadic downsampling, producing two output vectors that contain namely approximation (i.e., cA) and detail (i.e., cD) coefficients. The first convolution applies a low-pass filter g , the second one a high-pass filter h related to g in a quadrature relationship as they derive from the same mother wavelet. The filter coefficients are pre-computed and passed to the algorithm as input parameters. Initial input data are also provided for the first level, while approximation coefficients represent the input of the next level. The filter size (FS) is an even number equal or greater to two, and the case of FS=2 is also referred to as *Haar wavelet*.

In our design, we applied three main optimizations. First, we implemented a *strided convolution* routine that performs convolution and downsampling of both filters in a single step, reducing the total number of instructions required to compute cA and cD . Second, we provided a coding variant for the Haar wavelet, which does not require border paddings and fully unrolls the last loop to compensate for the small filter size that induces memory access stalls. Third, the algorithm copies cA values into the input data structure at the end of the second loop, reducing the total memory footprint for data allocation (e.g., GSL requires an additional memory buffer).

We applied loop-level parallelization on the second level. In the general code variant, this level is further split into three parts, corresponding to the border and inner data. The size of the iteration space for the border computation is equal to $FS-1$; consequently, an ideal workload balancing of this code is impracticable when there are more numerous cores than iterations. Synchronization barriers are required after applying the filters and after preparing input data for the next level.

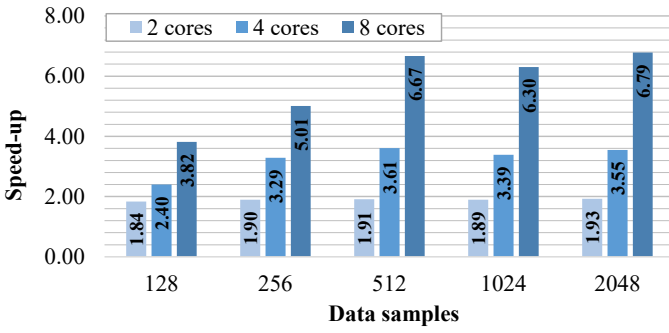


Fig. 2. Speed-up of STFT varying the number of cores and the input size.

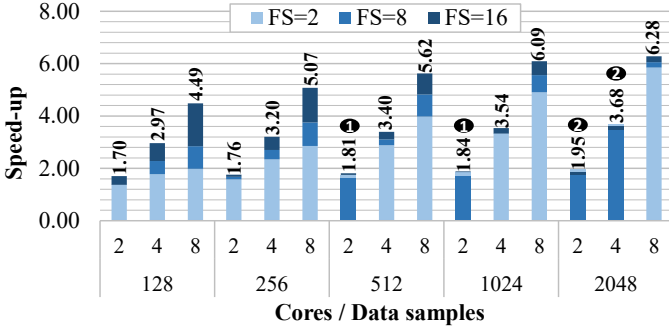


Fig. 3. Speed-up of DWT varying the number of cores, the input size, and the filter size (FS).

III. EVALUATION

We performed an experimental evaluation using a cycle-accurate PULP emulator implemented on a Xilinx UltraScale+ VCU118 FPGA board¹. We used a configuration with 8 cores and 4 FPUs, which is the most energy-efficient for near-sensor applications [17]. We measured execution cycles, instructions, and stalls for each code region employing hardware performance counters. In addition, we performed a power analysis with Synopsys PrimeTime 2019.12, considering a nominal voltage of 0.65 V and a frequency of 250 MHz. The metrics of interest for our analysis are *parallel speed-up* (sequential execution time over parallel execution time), *throughput* (number of input data samples over total execution cycles) and *energy efficiency* (operations performed in a second over power consumption).

A. Overheads and parallel speed-up

Parallel performance is limited by overheads deriving from two main sources: stalls in the core pipeline during the execution of instructions and time spent in synchronization. Table I reports stalls and synchronization occurrences considering 8 cores and 2048 data samples. This table also reports the throughput of the algorithms as an absolute performance metric. On PULP, pipeline stalls derive from memory latency (load-use stalls), concurrent accesses to the TCDM banks (memory contention arbitration stalls), concurrent requests to a shared FPU (FPU contention arbitration stalls), and instruction cache misses. Analyzing the cause of stalls provided guidance for fine-grain tuning of the optimization techniques described in the previous section. Synchronization happens on barriers required by the algorithms. As reported in [15], the barrier

TABLE I
INSTRUCTIONS, HARDWARE STALLS, SYNCHRONIZATIONS OCCURRENCES, AND THROUGHPUT (EXECUTION ON 8 CORES, 2048 DATA SAMPLES).

	STFT	DWT		
		FS=2	FS=8	FS=16
Instructions (per core)	20677	3090	12629	23848
TCDM stalls [cycles]	1034	58	610	1539
I-cache stalls [cycles]	504	127	76	189
FPU stalls [cycles]	4992	517	2285	3022
Synchronization occurrences	11	20	20	20
Throughput [samples/ μ s]	18.81	134.35	32.78	17.86

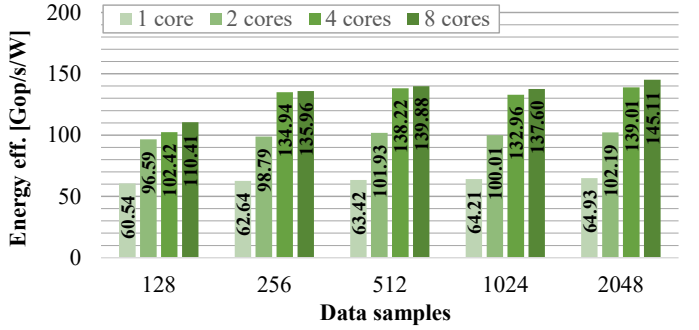


Fig. 4. Energy efficiency of STFT varying number of cores and input size.

cost with the event unit is 6 cycles, which implies an average synchronization overhead of around 1.41%. Without the event unit, the barrier overhead is 176 cycles (on 8 cores), and the average synchronization overhead rises to 41.48%.

Fig. 2 reports the parallel speed-up of STFT. Each value on the x-axis corresponds to a fixed number of input data samples, while the window overlap does not affect speed-up. In general, the speed-up increases with data samples since this trend amortizes the overheads due to loop-level parallelism. The case of 512 data samples is out of trend because a preliminary radix-2 stage is not required since 512 is a power of eight. Table I shows that the 8-cores configuration is mainly limited by the FPU sharing since the contribution of FPU stalls (4992) over the total instructions (20677) is around 25%.

Fig. 3 reports the parallel speed-up of DWT. Each bar provides the values of the speed-up for a filter size (FS) equal to 2 (light shade), 8 (intermediate shade), and 16 (dark shade). In general, the speed-up increases with the filter size, but there are some remarkable exceptions. Executing on 2 cores with a workload of size 512 or 1024, the speed-up of FS=2 is higher than FS=8 (label 1). This effect is even more evident when executing a workload of 2048 data samples on 2 or 4 cores, where the case of FS=2 becomes the highest speed-up (label 2). This trend is because the parallel orchestration of the Haar wavelet is more lightweight, as explained in Section II-B. The 8-core configuration implies additional overheads (i.e., TCDM contentions and FPU stalls in Table I) and workload unbalancing (see Section II-B) hiding this beneficial effect.

B. Energy efficiency

Fig. 4 and Fig. 5 depict the energy efficiency of STFT and DWT, respectively. This metric grows with the input size when considering a fixed number of cores. Moreover, it increases by fixing the input size and changing the number of cores from 1

¹<https://github.com/pulp-platform/pulp/tree/master/fpga/pulpassimo-zcu104>

TABLE II
COMPARISON WITH GSL AND KISS FFT ON 8-CORE PULP (CYCLES).

Our FFT	Kiss FFT	Our DWT (FS=2)	GSL DWT (FS=2)
27218	2293231	3881	67973

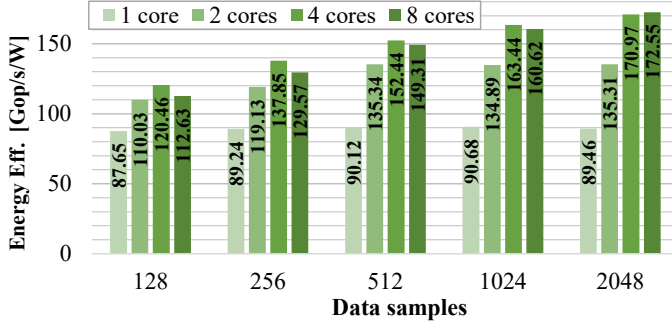


Fig. 5. Energy efficiency of DWT varying number of cores and input size.

to 4, but it presents a trend inversion passing from 4 to 8 cores in DWT. Again, this is due to the additional overheads implied by the 8 cores configurations, together with the higher power consumption. This effect can be amortized by computing a bigger input set. For instance, the energy efficiency is almost equivalent between the 4-core and 8-core configurations for 2048 samples.

C. Comparison with other libraries and architectures

Table II compares the execution time of our solution with the ones of the libraries introduced in Section I, namely GSL and Kiss FFT, executing on an 8-core PULP cluster with an input size of 2048 samples. GSL only supports Haar wavelets (FS=2). We applied minimal modifications to these algorithms to use the PULP HAL. Overall, our design outperforms other libraries, thanks to our domain-specific code optimizations and parallel design. The design of Kiss FFT applies parallelization only at the outer loop level (transform steps), which does not guarantee a perfect workload balancing when the number of cores is higher than four, with detrimental effects on performance. Both algorithms do not employ the optimization techniques (e.g., loop unrolling) described in Section II.

We also performed a comparison between PULP (8-core configuration) and Cortex-M4, using an STM32F401C-DISCO development board running at 1.7 V and 84 MHz, with an average power consumption of 20 mW. The STFT implementation for the Cortex-M4 platform makes use of the `arm_rfft_fast_f32` function from CMSIS-DSP, which is partially written using inline assembly and is the most efficient FFT implementation available for this platform. The DWT implementation for Cortex-M4 uses our library since a preliminary analysis highlighted that it is 60% faster than GSL. Fig. 6 shows that parallel execution on the PULP platform outperforms Cortex-M4 by one order of magnitude.

In terms of energy efficiency, PULP achieves 145.11 and 172.55 Gop/s/W for STFT and DWT, respectively, as reported in Figs. 4 and 5. Considering the performance measured on Cortex-M4 (71.4 and 75.6 Mop/s), it reaches 3.42 and 3.63 Gop/s/W, about two orders of magnitude worse than PULP.

IV. CONCLUSION

In this letter, we propose an optimized design for STFT and DWT algorithms targeting parallel ULP IoT end nodes.

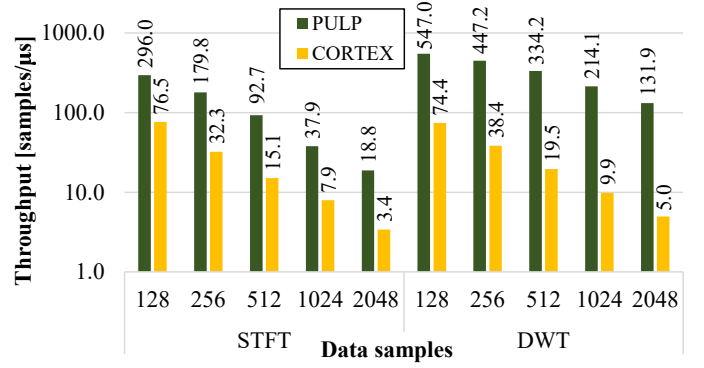


Fig. 6. STFT and DWT throughput on PULP / Cortex-M4 varying input size.

Experimental results assess that both algorithms achieve high parallel speed-ups, throughput, and energy efficiency on the PULP platform, outperforming a conventional single-core MCU in terms of performance and energy efficiency. These algorithms are pervasive in many applications running on IoT end nodes. For this reason, high optimization is crucial to satisfy the ever-increasing requirements of future applications.

REFERENCES

- [1] E. Flamand, *et al.*, "GAP-8: A RISC-V SoC for AI at the Edge of the IoT," in *IEEE 29th Int. Conf. on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2018, pp. 1–4.
- [2] C. Chen *et al.*, "Xuantie-910: A Commercial Multi-Core 12-Stage Pipeline Out-of-Order 64-bit High Performance RISC-V Processor with Vector Extension: Industrial Product," in *ACM/IEEE 47th Annual Int. Symp. on Comp. Arch. (ISCA)*, 2020, pp. 52–64.
- [3] D. Rossi *et al.*, "PULP: A parallel ultra low power platform for next generation IoT applications," in *2015 IEEE Hot Chips 27 Symposium (HCS)*. IEEE, 2015, pp. 1–39.
- [4] M. Alioto, E. Consoli, and G. Palumbo, *Flip-Flop Design in Nanometer CMOS*. Springer, 2016.
- [5] A. Pullini *et al.*, "Mr.Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, 2019.
- [6] CMSIS source code. Website: https://github.com/ARM-software/CMSIS_5. Accessed: 2020-08-20.
- [7] S. Benatti, E. Farella, and L. Benini, "Towards EMG control interface for smart garments," in *Proc. of the 2014 ACM Int. Symposium on Wearable Computers: Adjunct Program*, 2014, pp. 163–170.
- [8] E. Cabal-Yepez, A. G. Garcia-Ramirez *et al.*, "Reconfigurable monitoring system for time-frequency analysis on industrial equipment through STFT and DWT," *IEEE Trans. on Industrial Informatics*, vol. 9, no. 2, pp. 760–771, 2012.
- [9] X. Wang, M. Magno, L. Cavigelli, and L. Benini, "FANN-on-MCU: An open-source toolkit for energy-efficient neural network inference at the edge of the internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4403–4417, 2020.
- [10] F. Montagna, S. Benatti, and D. Rossi, "Flexible, Scalable and Energy Efficient Bio-Signals Processing on the PULP Platform: A Case Study on Seizure Detection," *Journal of Low Power Electronics and Applications*, vol. 7, no. 2, p. 16, 2017.
- [11] M. Frigo *et al.*, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.
- [12] Kiss FFT. <https://github.com/mborgerding/kissfft>. Accessed: 2020-08-20.
- [13] GLS library. <https://github.com/ampl/gsl>. Accessed: 2020-08-20.
- [14] PULP Software Development Kit. Website: <https://github.com/pulp-platform/pulp-sdk>. Accessed: 2020-08-20.
- [15] F. Glaser *et al.*, "Energy-Efficient Hardware-Accelerated Synchronization for Shared-L1-Memory Multiprocessor Clusters," *IEEE Trans. on Parallel and Distributed Systems*, vol. 32, no. 03, pp. 633–648, 2021.
- [16] S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, 1989.
- [17] F. Montagna *et al.*, "A transprecision floating-point cluster for efficient near-sensor data analytics," *arXiv preprint arXiv:2008.12243*, 2020.