

This is the peer reviewed version of the following article:

A Low-Power Transprecision Floating-Point Cluster for Efficient Near-Sensor Data Analytics / Montagna, F.; Mach, S.; Benatti, S.; Garofalo, A.; Ottavi, G.; Benini, L.; Rossi, D.; Tagliavini, G.. - In: IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. - ISSN 1045-9219. - 33:5(2022), pp. 1038-1053. [10.1109/TPDS.2021.3101764]

Terms of use:

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.

26/04/2024 05:24

(Article begins on next page)

A Transprecision Floating-Point Cluster for Efficient Near-Sensor Data Analytics

Fabio Montagna, Stefan Mach, Simone Benatti, Angelo Garofalo, Gianmarco Ottavi, Luca Benini, *Fellow, IEEE*, Davide Rossi, *Member, IEEE*, and Giuseppe Tagliavini, *Member, IEEE*

Abstract—Recent applications in the domain of near-sensor computing require the adoption of floating-point arithmetic to reconcile high precision results with a wide dynamic range. In this paper, we propose a multi-core computing cluster that leverages the fined-grained tunable principles of transprecision computing to provide support to near-sensor applications at a minimum power budget. Our design – based on the open-source RISC-V architecture – combines parallelization and sub-word vectorization with near-threshold operation, leading to a highly scalable and versatile system. We perform an exhaustive exploration of the design space of the transprecision cluster on a cycle-accurate FPGA emulator, with the aim to identify the most efficient configurations in terms of performance, energy efficiency, and area efficiency. We also provide a full-fledged software stack support, including a parallel runtime and a compilation toolchain, to enable the development of end-to-end applications. We perform an experimental assessment of our design on a set of benchmarks representative of the near-sensor processing domain, complementing the timing results with a post place-&route analysis of the power consumption. Finally, a comparison with the state-of-the-art shows that our solution outperforms the competitors in energy efficiency, reaching a peak of 97 Gflop/s/W on single-precision scalars and 162 Gflop/s/W on half-precision vectors.

Index Terms—RISC-V, multi-core, transprecision computing, near-sensor computing, energy efficiency



1 INTRODUCTION

THE pervasive adoption of edge computing is increasing the computational demand for algorithms targeted on embedded devices. Besides the aggressive optimization strategies adopted on the algorithmic side [1], there is a great effort to find the best trade-off between architectural features and computational capabilities [2]. Indeed, deploying artificial intelligence algorithms or digital signal processing (DSP) on near-sensor devices poses several challenges to resource-constrained low-power embedded systems.

Fixed-point arithmetic is a well-established paradigm in embedded systems optimization since it allows a simplified numerical representation for real numbers at high energy efficiency [3]. Nevertheless, many applications require high precision results characterized by a wide dynamic range (e.g., the accumulation stage of support vectors, or feed-forward inference for deep neural networks). In these cases, fixed-point implementations may suffer from numerical instability, requiring an in-depth analysis to make the result reliable and additional code sections to normalize and adjust the dynamic range avoiding saturation (e.g., the fixed-

point implementation of linear time-invariant digital filters described in [4]). As a result, fixed-point arithmetic is not necessarily the most energy-efficient solution since the code requires real-time adaptations of the dynamic range that affect performance significantly and increase the time-to-market [3]. To cope with these issues, the adoption of single-precision floating-point (FP) arithmetic is a well-established paradigm for embedded low-power systems, such as ARM Cortex M4, a microcontroller (MCU) architecture that is the *de facto* standard for FP-capable low-power edge nodes. Combining FP and fixed-point arithmetic, depending on the computational requirements, is the typical approach for optimizing Cortex-M4 applications. The main shortcomings of this approach are related to the manual analysis for the format selection (float vs. fixed), the tuning required for adjusting the fixed-point dynamic range, and the software overhead to make the format conversions. Furthermore, usage of mixed (i.e., floating-fixed point) representation introduces several architectural bottlenecks in managing the pipelines and the register file, such as flushing or stalls that reduce the computational efficiency of these approaches. Finally, at least in commercial architectures, the floating-point unit (FPU) cannot be turned off while the core is executing fixed-point operations, resulting in a further reduction of energy efficiency.

In this scenario, *transprecision computing* [5] is emerging as a successful paradigm for embedded computing systems. This paradigm is an evolution of approximate computing, and it aims at tuning approximation at a fine grain during the computation progress through hardware and software control mechanisms. In the context of FP computations, this approach requires the availability of hardware units providing efficient support for multiple FP formats. The

- F. Montagna and G. Tagliavini are with the Dept. of Computer Science and Engineering (DISI), University of Bologna, Italy.
E-mail: {fabio.montagna, giuseppe.tagliavini}@unibo.it
- S. Mach and L. Benini are with the Dept. of Information Technology and Electrical Engineering (D-ITET), ETH Zürich, Switzerland.
E-mail: {smach, benini}@iis.ee.ethz.ch
- A. Garofalo, G. Ottavi, S. Benatti, L. Benini and D. Rossi are with the Dept. of Electrical, Electronic, and Information Engineering (DEI), University of Bologna, Italy.
E-mail: {angelo.garofalo, gianmarco.ottavi2, simone.benatti, davide.rossi, luca.benini}@unibo.it

This work has been partially supported by the European Unions Horizon 2020 research and innovation programme under grant agreement numbers 732631 (OPRECOMP) and 857191 (IOTWINS).

TABLE 1
Floating-point formats used in low-power embedded systems.

Format ¹	Exponent	Mantissa	Range	Accuracy ²
<i>float</i>	8	23	$1.2 \times 10^{-38} - 3.4 \times 10^{38}$	7.2
<i>bfloat16</i>	8	7	$1.2 \times 10^{-38} - 3.4 \times 10^{38}$	2.4
<i>float16</i>	5	11	$5.9 \times 10^{-88} - 6.5 \times 10^4$	3.6

¹ Number of bits. ² Decimal digits.

IEEE 754-2008 standard [6] describes five FP formats, and two are suitable for our target: the 16-bits half-precision format (*float16*) and the 32-bits single-precision format (*float*). Moreover, *bfloat16* is an alternative 16-bit format dedicating 8 bits to the exponent field, in contrast with the 5 bits used in the IEEE half-precision format; this trade-off allows to handle the same dynamic range of the *float* format losing some precision. Relevant applications in the field of near-sensor computing [1] as well as state-of-the-art (SoA) machine learning algorithms widely rely on *float16* and *bfloat16* formats since many ML models, such as Temporal Convolutional Networks (TCN) [7] or Convolutional Neural Networks (CNN) [8], tolerate lower precision arithmetic without losing their accuracy [9]. Adopting the smaller format that satisfies the application accuracy requirements paves the way for substantial improvements in performance and energy consumption. Still, programmers need full software support in the compilation toolchain and also a consolidated methodology for tuning the precision of FP variables [10]. Table 1 provides an overview of these formats.

Lowering the bitwidth of FP variables paves the way for crucial optimizations. The most relevant one is the possibility of performing instructions on multiple sub-word elements simultaneously using a single-instruction-multiple-data (SIMD) approach on 16-bits vectors. These data types are known as *packed-SIMD vectors*. SIMD operations act on multiple data elements of the same size and type simultaneously, offering a theoretical $2\times$ speed-up for 16-bits data. Moreover, vectorization of 16-bits types enables an equivalent reduction of the memory footprint, allowing to store bigger models in the same amount of memory. This approach also enables more effective data movements, as multiple elements can be transferred concurrently.

An architectural design that aims to achieve the goals discussed above must exploit additional features of the ultra-low-power (ULP) computing domain. A tightly-coupled cluster composed of several processing elements (PEs) enables to improve the computational capabilities of the system using parallel programming techniques, without increasing the operating frequency. Specialized hardware extensions allow programmers to accelerate key parallel patterns and exploit the advantages of packed-SIMD operations. Combining these features with near-threshold computing on a fully programmable multi-core architecture leads to a highly scalable and versatile system suitable for a wide range of applications. The number of FPUs and the sharing factor among the cores of the cluster require a careful evaluation since these aspects directly impact area and energy efficiency. For instance, having a dedicated FPU for the cores can be detrimental if the data demand from the PEs can not be satisfied by the memory throughput: this effect is

known as the Von Neumann bottleneck. Thus, reducing the number of FPUs and adopting a sharing policy among the cores can be beneficial to improve the system’s efficiency. Another aspect to consider is the pipelining of the FPU unit, which allows designers to increase the maximum operating frequency to the cost of a deterioration of the performance. In the depicted scenario, finding the best trade-off requires an accurate exploration of the design space that includes the definition of adequate metrics and an experimental assessment on kernels from end-to-end applications.

In this paper, we propose the design of a *transprecision computing cluster* tailored for applications in the domain of near-sensors computing. Our work includes the architectural design of the transprecision cluster and the full specification of FPU and interconnects. We also provide full-fledged software support for the transprecision cluster, including a runtime environment for parallel programming and an extended compilation toolchain. We performed a comprehensive exploration of the design space considering different architectural configurations of the transprecision cluster: the number of cores, the number of FPUs and the related sharing factor, the number of pipeline stages in the FPUs. We have performed this exploration on an accurate hardware emulator implemented on an FPGA board. We have performed experiments on a set of benchmarks, including kernels commonly used in end-to-end applications identifying the most efficient solutions in terms of performance, power, and area; for this evaluation, we have considered the experimental results and power/area figures derived from post place-&-route (P&R) models in 22nm FDX technology. Based on this exploration, we derive guidelines to instantiate an optimal proper cluster configuration depending on the target application domain and expected performance. Finally, we have compared the most efficient configurations deriving from the design space exploration with SoA solutions, considering a broader scenario that includes high-performance and embedded computing domains.

Our experimental results show that the configuration with 16 cores and private FPUs configured with one pipeline stage provide the best performance (5.92 Gflop/s), the one with 16 cores and private FPUs configured with zero pipeline stages is the most energy-efficient (167 Gflop/s/W), and finally the configuration with 8 cores and 4 shared FPUs configured with one pipeline stage is the most area-efficient (3.5 Gflop/s/mm²). Finally, the energy efficiency of the transprecision cluster outperforms all the other solutions that provide FP support in the area of embedded computing.

The rest of the paper is organized as follows: Section 2 presents the related work. The proposed architecture of the transprecision cluster is presented in Section 3. Section 4 describes the programming model and the compilation toolchain. Sections 5 and 6 present the experimental results and a comparison with previous works, respectively. Finally, Section 7 concludes the whole paper.

2 RELATED WORK

2.1 Alternative formats

The IEEE754-2008 standard includes the definition of decimal formats for contexts where the accumulation of round-

ing errors in binary-radix numbers can lead to unacceptable accuracy losses. This effect is due to the fact that a set of finite radix-10 numbers becomes periodic when represented in radix-2 notation. Wahba et al. [11] present a solution reducing by 6% percent the latency of an FP decimal unit compared to SoA solutions, and saving 23% of the total area compared to solutions that include two FP units (for binary and decimal support, respectively). Decimal FPUs are characterized by a longer critical path and a larger area than binary units since representing a decimal digit requires four bits. For these reasons, they are not a suitable alternative for integration in our transprecision cluster.

In recent years, researchers have started to explore custom formats that are alternative to the IEEE standard ones and its closer derivatives. Gautschi et al. [12] propose an FPU based on the logarithmic number system (LNU), which is up to $4.1 \times$ more energy efficient than standard FP units in non-linear processing kernels. Universal numbers (*UNUMs*) [13] adopt a variable-width representation based on interval arithmetic to guarantee that the result contains the exact solution [14]. The variable width provided by UNUM enables to scale up precision in scientific computing applications [15], but hardware implementations not suitable for the area and energy constraints of the embedded computing domain. A recent version of the UNUM specification, known as *UNUM type III* or *posit* [16], proposes a solution to the hardware overhead issue. [14] introduces a posit arithmetic unit supporting additions and subtractions, coupled with dedicated compression units to reduce the memory footprint. Considering the overhead of current hardware implementations and their limited benefits (in [14], authors estimate a reduction of memory footprint around 7%), counterpoised with a significant effort in code refactoring, in this work we have not considered these hardware components as viable candidates for the transprecision cluster design.

2.2 Transprecision computing building blocks

The choice of an energy-efficient and transprecision-enabled FPU design is a key enabler for this work. In literature, there are several designs of FPUs that enable transprecision operations. For instance, Kaul et al. [17] describe a variable-precision fused multiply-and-add (FMA) unit with vector support (1, 2, or 4 ways). Their design considers 8 bits for the exponents and 24 bits for the mantissa. Moreover, a 5-bits certainty field tracks the number of accurate mantissa bits: Computations that do not fulfill the accuracy constraints provided by the application are recomputed with increased precision. The energy consumption for a 32 nm CMOS implementation is 19.4 pJ/FLOP, even though the overhead due to precision tracking and fixed-size exponents increases the total energy consumption at the application level. Moreover, if maximum precision is required, applications become very inefficient, due to the need for repeated operations performed at a lower precision.

Nannarelli [18] describes the design of an FPU based on the Tunable Floating-Point (TFP) format, which supports a variable number of bits for mantissa (from 4 to 24) and exponent (from 5 to 8). However, this solution does not support vectorization, which is a key enabler for energy efficiency. Jaiswal et al. [19] present a pipelined design of

two FP adders that support multiple precision configurations. The results are promising in terms of area and energy efficiency, but this solution does not support additional FP operations. Hardfloat [20] is an open-source library (written in Chisel) that contains parameterized blocks for FMA operations, conversions between integer and FP numbers, and conversions among different FP formats. At the current stage of development, this library offers individual function blocks instead of a fully-featured FPU, missing unit-level optimizations. Zhang et al. [21] present a multiple-precision FP FMA with vector support. Their work aims at minimizing the area overhead, but the hardware sharing inside the datapath constrains all formats to use the same latency. Moreover, the FPU does not provide any support for scalars in smaller formats.

FPnew [22] is an open-source transprecision floating-point unit (TP-FPU) capable of supporting a wide range of standard (double, float, and float16) and custom (bfloat16 and 8-bit minifloat) FP formats. FPnew supports both scalar and packet-SIMD operations, and the experimental results shown in [22] assess that this design outperforms all its competitors in terms of area and energy efficiency. We have integrated this FPU in our architecture, Section 3.2 describes its design and integration aspects in further detail. FPNew includes a DIVSQRT module to compute divisions and square roots using an iterative non-restoring divider, similar to the design presented in [23].

2.3 Vector units

Variable-length vector units have been initially introduced on the CRAY-1 architecture [24], and today they are a well-established solution in the high-end computer systems. The ARM Scalable Vector Extension (SVE) [25] is a vector extension introduces as a SIMD instruction set for the AArch64 architecture. The SVE specification allows system designers to choose a vector register length between 128 and 2,048 bits to satisfy different constraints. The programming model is vector-length agnostic; there is no need to recompile the source code or use compiler intrinsics to change the vector length. The A64FX chip by Fujitsu is realized in TSMC 7nm technology and implements the SVE extension, including 48 cores with support for 512-bit wide vectors and reaching peak performance of 2.7 Tflop/s [26]. This chip has been used in Fugaku, which entered the TOP500 list in June 2020 as the fastest supercomputer in the world.

The current working draft for the RISC-V V vector extension¹ defines a variable-length register file with vector operation semantics. This extension provides support for FP16, FP32, FP64, and FP128 types, and also includes widening FMA operations to support mixed-precision computations (e.g., multiplying two FP16 registers and adding the result to an FP32 register). ARA [27] and Hwacha [28] are two embodiments of this provisional standard. ARA includes an RV64 core and a 64-bit vector unit based on the version 0.5 draft of the RISC-V vector extension. Hwacha is based on the vector-fetch paradigm and is composed of an array of single-issue, in-order RISC-V Rocket cores [28]. In general, the area and power consumption of these solutions are too high for low-power, MCU-class processing systems. This

1. <https://github.com/riscv/riscv-v-spec>

observation is the main reason why vector semantics in ULP embedded systems are typically supported by providing packed-SIMD instructions.

2.4 Software-based transprecision approaches

Besides approaches involving custom HW design to enable mixed-precision operations, several researchers have proposed multiple-precision arithmetic libraries that extend the IEEE754 formats to perform FP computations with arbitrary precision. This solution allows application designers to overcome the limitations of fixed-format FP types without dedicated hardware support. ARPREC [29] and GNU MPFR [30] provide APIs to handle multiple formats characterized by a fixed-size exponent (a machine word) and an arbitrary size mantissa (multiples of a machine word). Arb [31] is a C library for arbitrary-precision interval arithmetic using the midpoint-radius representation that outperforms non-interval solutions such as MPFR in some applications. These libraries are widely used in contexts requiring high-dynamic range and are characterized by relaxed constraints on computation time and energy consumption (e.g., scientific computing on data center nodes). To speed up the library execution time, Lefèvre [32] presents a new algorithm to speed up the sum of arbitrary-precision FP number using the MPRF internal representation. However, the approach based on software emulation is not a viable solution for energy-efficient embedded systems, since both time and energy efficiency are negatively affected by at least an order of magnitude compared with solutions based on dedicated hardware.

Anderson et al. [33] propose a software approach for the reduced-precision representation of FP data. They define a set of non-standard floating-point multibyte formats (flytes) that can be converted to the next largest hardware type to perform arithmetic computations. The exponent is set to the maximum number of bits of the containing type to minimize the conversion overhead. The adoption of the vector units available on general-purpose processors (e.g., Intel Haswell) or high-end accelerators (e.g., Intel Xeon Phi) allows the software library to coalesce memory accesses and then amortize the conversion overhead.

2.5 Low-power parallel architectures for FP computing

Coarse Grain Reconfigurable Architectures (CGRAs) recently emerged as a promising solution for the near-sensor processing domain. CGRAs are systolic arrays containing a large number of processing elements with a low-latency routing interconnect. MuTARe [34] is a CGRA working in the near-threshold voltage regime to achieve low energy. This solution improves by 29% the energy efficiency of a heterogeneous platform based on the ARM big.LITTLE platform. However, MuTARe targets high-end embedded systems, and it does not provide FP support. Transpire [35] is a CGRA architecture with FP support. This architecture has a potential performance improvement of $10.06\times$ and can be $12.91\times$ more energy efficient than a RISC-V core extended with packed-SIMD vectorization. These benefits are due to the fact that the design of CGRAs enables programmers to exploit different combinations of data-level and

pipeline-based parallelism. However, the domain of near-sensor processing includes a wide variety of algorithms presenting complex access patterns that cannot be efficiently implemented on CGRAs.

Mr.Wolf [2] is a multi-core programmable processor implemented in CMOS 40nm technology. The platform includes a tiny (12 K gates) RISC-V core accelerated by a powerful 8-cores cluster of RI5CY cores [36] sharing two single-precision FPUs. The limited number of FPUs provided by this architecture represents a severe limitation to the maximum FP intensity that applications may expose. A primary contribution of our work consists of finding the best tradeoff between the number of cores and the number of available FPUs, yet considering strict area and power constraints, and exploiting transprecision units to improve performance and execution efficiency. In Section 6, we include Mr.Wolf in our comparison with SoA platforms.

Helium² is an extension of the Armv8.1-M architecture targeting low-power MCU-class computing systems. The ISA extension includes a set of scalar and vector instructions supporting fixed-point (8-bit, 16-bit, and 32-bit) and FP (float and float16, optionally double) formats. These instructions are beneficial for a wide range of near-sensor applications, from machine learning to DSP. The Cortex-M55³ core includes the Helium extension, but chips based on this IP are not yet available on the market to perform a comparison with our solution.

2.6 High-end embedded systems for FP computing

The most widely used commercial architectures for compute-intensive FP workloads are GP-GPUs. With the growth of emerging applications such as training of neural networks, they have also started to support reduced precision floating-point formats such as *brain-float* and *binary16*. Indeed, training algorithms for deep neural networks such as backpropagation is naturally robust to errors. These features of modern GPUs have also been exploited in other application domains, such as machine learning [37] and linear algebra [38], demonstrating significant benefits for performance and efficiency. NVidia Pascal has been the first GPU supporting 16-bit FP formats. NVidia Pascal features SIMD float16 operations, that can be executed using a single paired-operation instruction. Furthermore, the new Volta micro-architecture further extends support to reduced precision types featuring mixed-precision multiply-and-add instructions.

Other research works, targeting neural network training and FP intensive workloads, leverage more specialized architectures. Neurostream [39] is a streaming memory-mapped co-processor targeting inference and training of deep neural networks in near-memory computing systems. This design removes the register-file bottleneck of SIMD architectures accessing the memory exploiting programmable hardware loops and address generators, and enabling execution efficiency close to one MAC per cycle. Neurostream achieves an average performance of 240 Gflop/s within a power-budget of 2.5 W. The architecture has been further

2. <https://www.arm.com/why-arm/technologies/helium>

3. <https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m55>

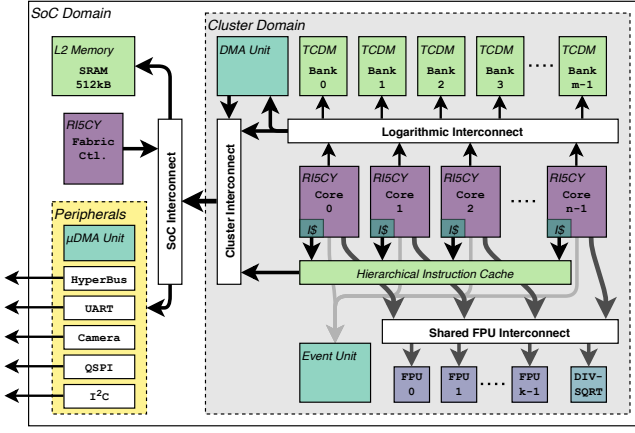


Fig. 1. Top-level view of the proposed transprecision cluster.

improved in [40], with a $2.7\times$ energy efficiency improvement over GPGPUs at $4.4\times$ less silicon area, delivering 1.2 TFLOP/s. The latter architecture has been implemented in 22nm FDX in Kosmodrom [41]. The chip includes two Ariane cores and one NTX accelerator. Kosmodrom achieves an energy efficiency of 260 Gflop/s/W and a 28 Gflop/s performance within a 6.2 mW to 400 mW power envelope.

In [42], the memory-mapped control has been replaced by a tiny general-purpose processor meant to drive double-precision FPUs, improving efficiency and flexibility of previous approaches. This architecture introduces two ISA extensions to reduce the pressure on the core: the stream semantic registers (SSR) and the floating-point repetition instruction (FREP). SSRs allow the core to implicitly encode memory accesses as register reads/writes, removing a significant number of explicit memory instructions. The FREP extension decouples the FP and integer pipeline by sequencing instructions from a micro-loop buffer. The evaluation on an octa-core cluster in 22 nm technology reports a $5\times$ multi-core speed-up and a $3.5\times$ gain in energy efficiency.

The architectures discussed in this section target the domain of servers and high-end embedded systems, and presenting further details is beyond the scope of our work. However, the comparison with these solutions provides useful insight and is discussed in Section 6.

3 ARCHITECTURE AND IMPLEMENTATION

3.1 Cluster Architecture

The cluster architecture proposed in this work is a soft IP implementing a tightly-coupled cluster of processors built around a parametric number of RISCY [36] cores. Fig.1 shows the top-level design of the transprecision cluster.

RISCY is a RISC-V based processor implementing a 4-stage in-order single-issue pipeline, supporting the RV32IMC instruction set and dedicated extensions for DSP and machine learning workloads [36]. The cores fetch instructions from a 2-levels shared instruction cache optimized for performance and energy efficiency when running SIMD workloads typical of near-sensor data analytics applications. To enable the single-cycle exchange of data among cores, they share a multi-banked Tightly-Coupled Data Memory (TCDM) behaving as a scratchpad memory

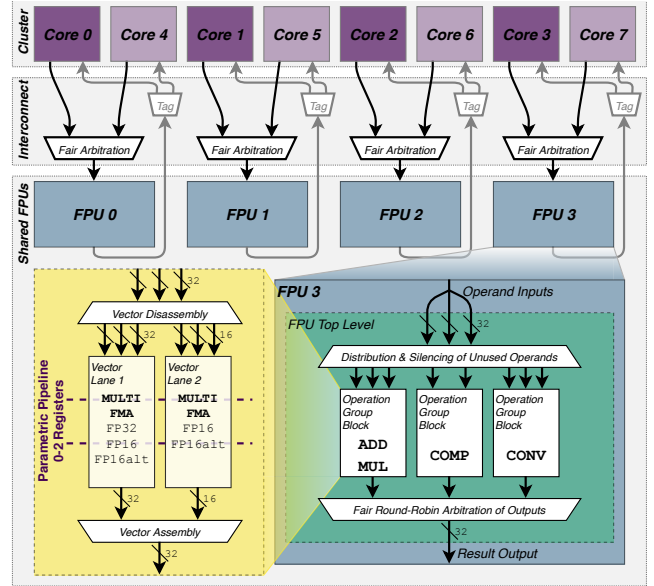


Fig. 2. FPU sharing for the 8-core, 4-FPU configuration.

(i.e., there is no data caching mechanism to avoid coherency and control overheads). The TCDM enables the cores to share data through a word-level interleaved, single-cycle latency logarithmic interconnect, allowing the execution of data-parallel programming models such as OpenMP. A dedicated hardware block (Event Unit) provides low-overhead support for fine-grained parallelism, accelerating the execution patterns typical of data-parallel programming models (e.g., thread dispatching, barriers, and critical regions) and enabling the adoption of power-saving policies when cores are idle [43].

Outside the cluster, at the SoC level, the architecture features one more memory hierarchy level, composed of a 15-cycle latency multi-banked scratchpad memory used to serve the data bus of the cores, the instruction cache refills, and the cluster DMA. We base the explorations performed in this work on a set of cluster configurations with 8 and 16 cores. The L2 memory comprises 512 kB, the TCDM is 64 kB for the 8-core configurations and 128 kB for the 16-core ones. The cluster cores are connected to multiple FPU instances, whose number and interconnect are a central part of our exploration. Unlike the standard configuration for the RISCY core, the proposed cluster does not employ core-private FPUs. Instead, a set of FPUs is shared among all cores in the cluster, using an interconnect which enables various mappings of cores to available FPUs. The next section provides insights into the FPU subsystem proposed in this work.

3.2 FPU and interconnect

The cluster exploits configurations of FPnew [22] as FPU instances in our evaluation. FPnew is a parametric FPU architecture that supports multiple FP formats, SIMD vectors, and the insertion of any number of pipeline stages. Fig. 2 (bottom) shows an architectural overview of a single shared FPU instance. The IP supports the standard IEEE formats, *binary32* (float) and *binary16* (float16), as well as *bfloat16*. Some operations such as Multiplication and Fused

Multiply-Add (FMA) can also be performed as multi-format operations, taking the product of two 16-bit operands but returning a 32-bit single-precision result. Such multi-format operations are helpful in many near-sensor data analytics applications accumulating data in a higher-precision variable to avoid overflows or losses of precision. To make full use of the 32-bit data path, we enable packed-SIMD operations for the 16-bit types, boosting the execution performance when using 16-bit data types. Division and square root operations are disabled in the FPU instances as these operations reside in stand-alone blocks (DIV-SQRT), which are shared separately. The DIV-SQRT units feature a fixed latency of 11, 7, and 6 cycles for *float*, *float16*, and *bfloat16*, respectively. Moreover, since DIV-SQRT is designed as an iterative block, back to back pipelined operations are not possible when using these units.

The individual FPU instances are linked to one or more cores through a logarithmic tree interconnect, allowing to share one FPU among multiple cores in a fully transparent way from a software perspective. On the core side, the interface of the interconnect replaces the FPU in the execution stage, mimicking a core-private unit. The FPU instances connect to the cores through an auxiliary processing unit (APU) interface, featuring a *ready/valid* handshake and support tagging of all in-flight operations, requiring no modification to be shared.

In the proposed design, we employ a partial interconnect with a static mapping of FPUs to cores, such that a core (or a group of cores) will always access the same physical FPU instance. It arbitrates cases of simultaneous accesses to the FPU by using a fair round-robin policy and propagating the ready signal to only one core, stalling other competing cores. As such, the fact that FPUs are shared is transparent to both the core and FPU instances. Moreover, we use a connection scheme with interleaved allocation to reduce access contentions on the FPUs in unbalanced workloads. For example, in a configuration featuring eight cores and four FPUs, units 0, 1, 2, 3 are shared among cores 0 & 4, 1 & 5, 2 & 6, and 3 & 7, respectively, as shown in Fig. 2 (top). This approach reduces the area and timing overhead compared to a monolithic, fully connected crossbar, which puts significant pressure on the paths from the cores to the first pipeline stage of the FPU, severely limiting the cluster’s operating frequency and jeopardizing energy efficiency. Moreover, it provides an almost optimal allocation (only up to 1% overhead in performance has been measured against a fully connected crossbar) avoiding contentions on the shared units also when the number of workers in parallel sections is smaller than the number of cores.

In the remainder of the paper, we present a design space exploration of the proposed transprecision cluster, modifying the key configuration parameters presented previously in this section, namely the pipeline stages and sharing factor. The rationale for the former lies in the fact that in most near sensor-data analytics applications, the density of FPU instructions is smaller than 50%, hence employing a private, per core FPU may form a bottleneck for area and energy. On the other hand, the pipelining of the FPU provides a powerful knob to tune the performance and energy efficiency of the transprecision cluster. If the number of cores and FPUs is equal (1/1 sharing factor), the

TABLE 2
Description of the architectural configurations of the proposed transprecision cluster that compose the design space. Cluster (8-16-cores), FP units (2-16), and pipeline stages (0-2).

Mnemonic	Cluster	FP units	Pipeline Stages
8c2f0p	8-cores	2	0
8c2f1p	8-cores	2	1
8c2f2p	8-cores	2	2
8c4f0p	8-cores	4	0
8c4f1p	8-cores	4	1
8c4f2p	8-cores	4	2
8c8f0p	8-cores	8	0
8c8f1p	8-cores	8	1
8c8f2p	8-cores	8	2
16c4f0p	16-cores	4	0
16c4f1p	16-cores	4	1
16c4f2p	16-cores	4	2
16c8f0p	16-cores	8	0
16c8f1p	16-cores	8	1
16c8f2p	16-cores	8	2
16c16f0p	16-cores	16	0
16c16f1p	16-cores	16	1
16c16f2p	16-cores	16	2

system effectively degenerates into a core-private scenario, and the interconnect disappears from the design. In all the considered configurations, a single DIV-SQRT unit is shared among all cores. Finally, the proposed exploration involves designs of 8-core and 16-core clusters with supply voltages ranging from 0.65 V to 0.8 V to explore the whole design space in between energy-efficient and high-performance solutions.

3.3 Implementation

This section presents the physical implementation results and related explorations of the proposed cluster. Table 2 describes the 18 different configurations, given by the combination of the three architectural parameters (number of cores, number of FP units, and number of FPU pipeline stages), as described in the previous section. The various configurations of the clusters have been synthesized using Synopsys Design Compiler 2019.12, using LVT libraries from 22nm FDX technology from Global Foundries. Physical implementation has been performed with Cadence Innovus v19.10-p002_1, using both 0.65 V near-threshold (NT) and 0.8 V super-threshold (ST) corners. We considered all permutations of operating conditions for signoff: fast and slow process transistors, 125°C and -40°C temperatures, $\pm 10\%$ of the voltage supply, as well as optimistic and pessimistic parasitics. Power analysis has been performed with Synopsys PrimeTime 2019.12 using the nominal corners at 0.65 V and 0.8 V, extracting *value change dump* (VCD) traces through parasitic-annotated post-layout simulation of a 32-bit floating-point matrix multiplication performed using Mentor Modelsim 2008.06. Each configuration has been synthesized and implemented at its maximum operating frequency. In contrast, power consumption has been analyzed at the same operating frequency for all configurations (100 MHz) to guarantee a fair comparison.

Fig. 3, Fig. 4 and Fig. 5 show the frequency, the area, and the power consumption of the cluster configurations analyzed in this work at 100 MHz. In Fig. 3, we report the minimum, maximum, and median values of the frequencies

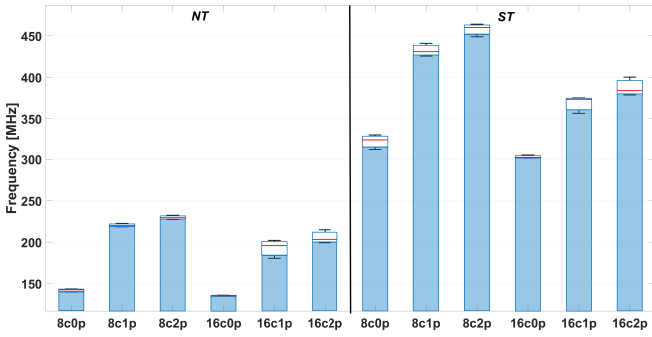


Fig. 3. Minimum, maximum, and median values of the frequencies for all the configurations of the transprecision cluster, divided in NT and ST voltage corners.

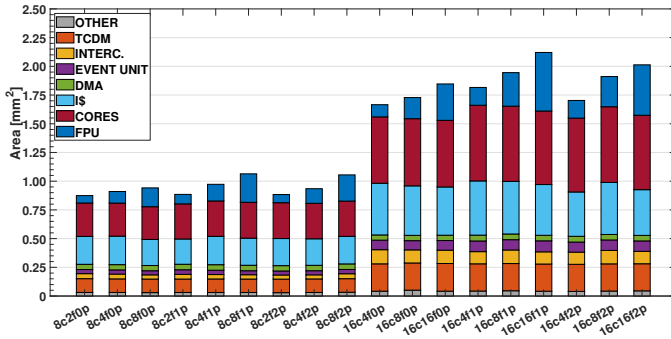


Fig. 4. Total area of all the configurations in the design space of the transprecision cluster.

obtained varying the number of FPUs. When considering single-cycle latency FPUs, we note that the entire system's operating frequency suffers profoundly. The long paths starting from the ID/EX registers of the core towards the FPUs and then back to the EX/WB registers form a considerable bottleneck for operating frequency. On the other hand, the absence of pipeline registers makes this solution quite small and low-power. When moving to single-stage pipeline solutions, we note a very significant increase in the operating frequency when using NT cells (almost 50%). In contrast, the performance increase using ST cells is more limited since the design already hits a structurally critical path from the TCDM SRAMs (featuring wide-voltage range but low-performance in ST) to the core through the logarithmic interconnect. In all configurations featuring one pipeline stage, we can observe an increase of power and area, due to the extra overhead of the additional pipeline stage. When adding a second pipeline stage to the FPUs, we can see another slight increase of frequency in all configurations. In these configurations, we also encounter structurally critical paths using NT, through control paths of the interconnect to the instruction cache. With two pipeline stages, although the area increases for all configurations, the power consumption tends to decrease thanks to the smaller timing pressure on the FPU.

Considering the sharing factor, we note that the impact on frequency caused by the FPU interconnect is negligible and that the area linearly increases when moving from

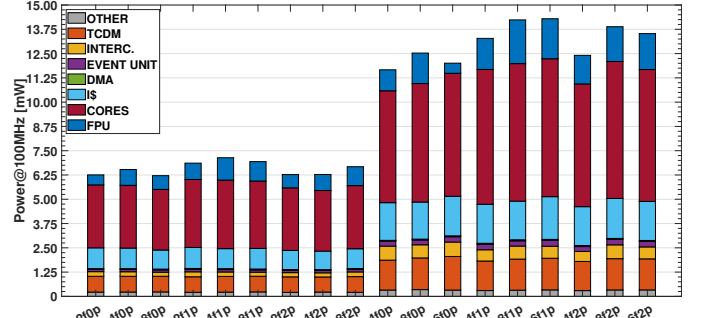


Fig. 5. Total power consumption (at 100 MHz) of all the configurations in the design space of the transprecision cluster.

1/4 to 1/1 sharing, for all configurations. On the other hand, when moving from 1/4 to 1/2 sharing factor, the power increases significantly due to the high utilization of the units. When moving from 1/2 to 1/1 sharing, we note that the power consumption decreases in almost all cases. This effect occurs because even if we consider a highly intensive benchmark (e.g., matrix multiplication), the FP intensity around 50% leads to underutilization of the available resources, causing smaller power consumption. Additionally, the 1/1 configuration removes the interconnect, which relaxes the paths through the FPU, leading to smaller power consumption. Finally, if we consider the scaling of the number of cores, we can notice that most of the power components scale linearly with the number of cores (i.e., core power, TCDM power, and FPU power). On the other hand, other components such as the interconnect and the instruction cache scale superlinearly, indicating a smaller efficiency for the 16-core configuration. Moreover, the operating frequency of the 16-core cluster decreases compared to the one using eight cores. This effect is due to the longer path through the interconnects. Finally, we can notice that the area increases less than linearly due to some blocks not being duplicated, such as the DMA, the event unit, and the shared banks of the IS.

4 PROGRAMMING MODEL AND COMPILATION TOOLCHAIN

The full exploitation of the transprecision cluster proposed in this work requires the support of a comprehensive software ecosystem, including a parallel programming model, vectorization techniques, and compiler support.

The architectural template of the transprecision cluster promotes a Single-Program Multiple-Data (SPMD) parallel paradigm. This paradigm is supported by a Hardware Abstraction Layer (HAL), which allows minimal access to the platform features. The HAL provides information such as the identifier of the core that can be used to organize the parallel workload for both data and task parallelism. In this programming model, all the cores of the cluster follow the same execution flow, unless the programmer explicitly indicates that a specific region should be executed by a subset of the cores, splitting the workload among the cores running concurrently on different data. Inter-core synchronization barriers are explicitly indicated to ensure the correctness

of the results. Our architecture features dedicated hardware support that allows optimizing synchronization construct like barriers or critical sections. The HAL layer provides the basic primitives to support high-level parallel programming models such as OpenMP. These models can provide a more intuitive interface at the cost of higher overhead; however, our exploration is focused on finding the maximum performance that we can obtain from real applications without considering a multi-layer software stack.

To provide compiler support, We have extended the RISC-V GCC toolchain with new data types (float16, bfloat16) and support for manual and automatic vectorization using packed-SIMD instructions. Programmers can explicitly use vector data types through a `typedef` declaration coupled with a `vector_size` attribute; the compiler automatically lowers standard arithmetic and comparison operations involving these types into their vector counterpart. The ISA extension also includes cast-and-pack operations that convert two scalar single-precision operands and insert them into two adjacent entries of a packed vector in the destination register. These operations aim at removing the bottleneck of “convert scalars and assemble vectors” operations that could seriously compromise performance and energy efficiency of transprecision computing techniques [1]. A set of compiler intrinsics provides access to cast-and-pack operations. The automatic vectorization pass of GCC operates on the middle-end intermediate representation⁴. This pass analyzes the loops to replace the scalar operations with the vectorial ones reducing the loop trip-count by the vectorization factor. We have also extended the standard GCC auto-vectorizer to use cast-and-pack operations. The original version only recognizes patterns involving multiple vector types with different widths using unrolling and vector-to-vector casts (i.e., cast-and-pack semantic was not supported).

In this work, we further extend the compiler back-end to support a parametric number of FPU pipeline stages. This parameter has a substantial impact on the instruction scheduling algorithm: imprecise modeling of the FPU instruction latency may introduce stalls due to data dependencies with the result. We have modified the FPU pipeline description to include the hardware functional units and introduce a command-line option to specify the number of stages in the target configuration. Based on this option, the model specifies different latency and reservation delay for the functional units involved in FP operations. Finally, we have a set of platform-specific parameters for the instruction scheduling algorithm. This algorithm uses a heuristic function to estimate the relative costs of operations; this value enables the choice of the best assembly sequence in case of multiple alternatives in the lowering process.

5 EXPERIMENTAL RESULTS

5.1 Experimental Set-up

The experiments have been performed on a hardware emulator implemented on a Xilinx UltraScale+ VCU118 FPGA

4. <https://www.gnu.org/software/gcc/projects/tree-ssa/vectorization.html>

TABLE 3

Main application domains (Domains), FP intensity (FP I.), and memory intensity (M. I.) for scalar and vector variants of the benchmarks.

Apps	Domains	Scalar		Vector	
		FP I.	M. I.	FP I.	M. I.
CONV	Audio, Image, ExG	0.33	0.67	0.28	0.29
DWT	Audio, Image, ExG	0.29	0.59	0.21	0.57
FFT	Audio, Image, ExG	0.32	0.52	0.26	0.38
FIR	Audio, Image, ExG	0.32	0.65	0.32	0.48
IIR	Audio, Image, ExG	0.19	0.55	0.17	0.33
KMEANS	ExG	0.55	0.36	0.44	0.30
MATMUL	Audio, Image, ExG	0.28	0.58	0.27	0.41
SVM	ExG	0.27	0.53	0.21	0.52

board⁵. The emulation on the FPGA provides cycle-accurate results, with a significant speed-up of the experiments compared to an RTL-equivalent simulation.

A set of non-intrusive per-core performance counters included in the hardware design record the number of executed instructions and cycles spent in different states (total, active, L2/TCDM memory stalls, TCDM contention, FPU stall, FPU contention, FPU write-back stall, instruction cache miss). We have generated all the bitstreams for all the configurations reported in Table 2 and, after loading a bitstream on the FPGA, we load and run application binaries using OpenOCD and GDB interfaces. The same interface is used to load a program binary in the L2 memory, start the program execution, and finally read the performance counters from an emulated terminal.

The values of power consumption used to calculate the efficiency have been derived from an annotated post-layout simulation, as described in Section 3.3.

5.2 Benchmarks

To evaluate the different configurations of the proposed transprecision cluster architecture, we analyzed eight benchmarks commonly used in the near-sensor processing applications for filtering, feature extraction, classification, and basic linear algebra functions. Table 3 illustrates the target benchmarks associated with their domains (i.e., audio processing, image processing, ExG biosignal processing).

The Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) are digital filters with various applications in data acquisition and analysis. The Discrete Wavelet Transform (DWT) is a standard kernel used for feature extraction, which decomposes a signal into a different level of frequency resolutions through a bank of Low Pass (LPF) and High Pass Filters (HPF), capturing both temporal and frequency information. The Fast Fourier Transform (FFT) is a mathematical method that transforms a signal from the time domain to the frequency domain. There are several variants of this algorithm; in this paper, we consider the decimation-in-frequency radix-2 variant. We consider a state-of-the-art supervised classifier, the Support Vector Machine (SVM), widely used in near-sensor applications [44]. We also include another classifier, named K-Means, which is an unsupervised ML algorithm able to inference an unknown outcome starting from input vectors. The last two kernels are basic linear algebra subprograms (BLAS)

5. <https://github.com/pulp-platform/pulp/tree/master/fpga/pulpiissimo-zcu104>

TABLE 4

Performance [$Gflop/s$], energy efficiency [$Gflop/s/W$], and area efficiency [$Gflop/s/mm^2$] executing the benchmarks on the 8-cores configurations. Performance and area efficiency are computed at 0.8 V, energy efficiency at 0.65 V. The last group of lines reports normalized average values. A box around the metric value highlights the best configuration for each benchmark.

		Scalar									Vector								
		8c2f0p	8c2f1p	8c2f2p	8c4f0p	8c4f1p	8c4f2p	8c8f0p	8c8f1p	8c8f2p	8c2f0p	8c2f1p	8c2f2p	8c4f0p	8c4f1p	8c4f2p	8c8f0p	8c8f1p	8c8f2p
CONV	PERF	1.16	1.62	1.74	1.43	1.94	1.97	1.52	2.04	1.96	1.91	2.57	2.19	2.04	2.89	2.36	2.32	2.98	2.35
	E. EFF	72	66	72	82	76	83	91	83	81	119	105	91	117	113	100	139	121	97
	A. EFF	1.5	1.8	2.0	1.8	2.0	2.1	1.6	1.9	1.9	2.5	2.9	2.5	2.6	3.0	2.5	2.5	2.8	2.2
DWT	PERF	0.54	0.73	0.77	0.70	0.87	0.86	0.75	0.95	0.85	0.83	1.12	1.18	0.89	1.17	1.19	0.92	1.21	1.16
	E. EFF	33	30	32	40	34	36	45	39	35	51	46	49	51	46	50	55	49	48
	A. EFF	0.7	0.8	0.9	0.9	0.9	0.9	0.8	0.9	0.8	1.1	1.3	1.3	1.1	1.2	1.3	1.0	1.1	1.1
FFT	PERF	0.67	0.91	0.97	0.92	1.23	1.21	1.02	1.37	1.27	1.21	1.56	1.54	1.49	1.83	1.66	1.60	1.98	1.63
	E. EFF	42	37	40	52	48	51	61	56	52	75	64	64	85	72	70	96	80	67
	A. EFF	0.9	1.0	1.1	1.2	1.3	1.3	1.1	1.3	1.2	1.6	1.8	1.7	1.9	1.9	1.8	1.7	1.9	1.5
FIR	PERF	1.21	1.54	1.40	1.49	1.76	1.48	1.62	1.88	1.47	2.24	3.03	2.76	2.54	3.38	2.86	2.70	3.57	2.79
	E. EFF	75	63	58	85	69	63	97	76	61	139	124	114	145	132	121	162	145	115
	A. EFF	1.6	1.7	1.6	1.9	1.8	1.6	1.7	1.8	1.4	3.0	3.4	3.1	3.2	3.5	3.1	2.9	3.3	2.6
IIR	PERF	0.61	0.82	0.86	0.70	0.90	0.91	0.74	0.94	0.91	1.06	1.40	1.46	1.15	1.49	1.49	1.19	1.55	1.48
	E. EFF	38	33	35	40	35	39	45	38	37	66	57	60	65	58	63	72	63	61
	A. EFF	0.8	0.9	1.0	0.9	0.9	1.0	0.8	0.9	0.9	1.4	1.6	1.6	1.4	1.5	1.6	1.3	1.5	1.4
K-M.	PERF	0.75	1.02	1.05	1.15	1.49	1.30	1.34	1.68	1.30	1.21	1.64	1.72	1.68	2.13	2.06	1.88	2.33	2.10
	E. EFF	47	42	43	66	58	55	80	68	53	75	67	71	96	83	87	113	94	86
	A. EFF	1.0	1.1	1.2	1.4	1.5	1.4	1.4	1.6	1.2	1.6	1.9	1.9	2.1	2.2	2.2	2.0	2.2	2.0
MAT.	PERF	1.06	1.54	1.51	1.28	1.74	1.61	1.35	1.81	1.61	2.10	2.77	2.68	2.36	3.16	2.77	2.46	3.32	2.71
	E. EFF	66	63	62	73	68	68	81	73	66	130	113	111	135	123	117	148	135	111
	A. EFF	1.4	1.7	1.7	1.6	1.8	1.7	1.4	1.7	1.5	2.8	3.1	3.0	3.0	3.2	3.0	2.6	3.1	2.6
SVM	PERF	0.53	0.69	0.69	0.59	0.74	0.71	0.62	0.77	0.70	0.63	0.85	0.82	0.68	0.89	0.83	0.69	0.91	0.81
	E. EFF	33	28	29	34	29	30	37	31	29	39	35	34	39	35	35	42	37	33
	A. EFF	0.7	0.8	0.8	0.7	0.8	0.8	0.7	0.7	0.7	0.8	1.0	0.9	0.9	0.9	0.9	0.7	0.9	0.8
NAVG	PERF	0.00	0.24	0.26	0.16	0.40	0.35	0.23	0.48	0.35	0.40	0.76	0.73	0.54	0.92	0.80	0.62	1.00	0.78
	E. EFF	0.13	0.01	0.04	0.27	0.12	0.16	0.43	0.24	0.13	0.73	0.54	0.52	0.79	0.62	0.62	1.00	0.76	0.56
	A. EFF	0.03	0.20	0.23	0.24	0.30	0.29	0.12	0.27	0.14	0.66	0.91	0.87	0.80	0.94	0.86	0.61	0.85	0.61

TABLE 5

Performance [$Gflop/s$], energy efficiency [$Gflop/s/W$], and area efficiency [$Gflop/s/mm^2$] executing the benchmarks on the 16-cores configurations. Performance and area efficiency are computed at 0.8 V, energy efficiency at 0.65 V. The last group of lines reports normalized average values. A box around the metric value highlights the best configuration for each benchmark.

		Scalar									Vector								
		16c4f0p	16c4f1p	16c4f2p	16c8f0p	16c8f1p	16c8f2p	16c16f0p	16c16f1p	16c16f2p	16c4f0p	16c4f1p	16c4f2p	16c8f0p	16c8f2p	16c16f0p	16c16f1p	16c16f2p	
CONV	PERF	2.19	2.80	2.93	2.61	3.10	3.14	2.71	3.37	3.26	3.51	4.28	3.63	3.69	4.54	3.74	4.00	4.78	3.87
	E. EFF	77	69	72	84	75	73	94	79	78	123	106	89	118	110	87	140	113	92
	A. EFF	1.5	1.8	2.0	1.7	1.8	1.9	1.7	1.8	1.8	2.5	2.7	2.5	2.5	2.7	2.2	2.5	2.5	2.2
DWT	PERF	0.74	0.91	0.95	0.87	0.98	0.97	0.89	1.06	1.00	0.88	1.07	1.10	0.94	1.05	1.06	0.94	1.11	1.08
	E. EFF	26	22	23	28	24	23	31	25	24	31	26	27	30	26	25	33	26	26
	A. EFF	0.5	0.6	0.6	0.6	0.6	0.6	0.5	0.6	0.6	0.6	0.7	0.8	0.6	0.6	0.6	0.6	0.6	0.6
FFT	PERF	1.21	1.51	1.56	1.52	1.78	1.81	1.60	1.99	1.90	2.13	2.54	2.50	2.25	2.62	2.53	2.22	2.74	2.58
	E. EFF	42	37	38	49	43	42	56	47	45	74	62	61	72	63	59	78	64	62
	A. EFF	0.8	1.0	1.1	1.0	1.0	1.1	1.0	1.1	1.1	1.5	1.6	1.7	1.5	1.5	1.5	1.4	1.5	1.5
FIR	PERF	2.29	2.66	2.38	2.71	2.86	2.39	2.85	3.08	2.47	4.17	5.19	4.62	4.62	5.38	4.54	4.79	5.92	4.62
	E. EFF	80	66	59	87	69	56	99	73	59	146	128	114	148	130	106	167	139	110
	A. EFF	1.6	1.7	1.6	1.8	1.7	1.4	1.8	1.6	1.4	2.9	3.3	3.2	3.1	3.1	2.7	3.0	3.1	2.6
IIR	PERF	0.78	0.95	0.99	0.81	0.93	0.95	0.81	0.98	0.97	1.37	1.69	1.73	1.42	1.62	1.65	1.41	1.71	1.68
	E. EFF	27	23	24	26	23	22	28	23	23	48	42	42	46	39	39	49	40	40
	A. EFF	0.5	0.6	0.7	0.5	0.5	0.6	0.5	0.5	0.5	1.0	1.1	1.2	1.0	0.9	1.0	0.9	0.9	0.9
K-M.	PERF	1.14	1.39	1.39	1.28	1.45	1.35	1.25	1.50	1.40	1.72	2.11	2.13	1.95	2.28	2.22	1.93	2.43	2.29
	E. EFF	40	34	34	41	35	32	43	35	33	60	52	52	63	55	52	67	57	54
	A. EFF	0.8	0.9	1.0	0.9	0.8	0.8	0.8	0.8	0.8	1.2	1.3	1.5	1.3	1.3	1.3	1.2	1.3	1.3
MAT.	PERF	1.96	2.57	2.41	2.23	2.65	2.41	2.30	2.86	2.50	3.98	4.83	4.57	4.34	5.14	4.48	4.42	5.47	4.57
	E. EFF	68	63	59	72	64	56	80	67	60	139	119	113	139	125	104	154	129	109
	A. EFF	1.4	1.6	1.6	1.5	1.5	1.4	1.4	1.5	1.4	2.8	3.0	3.1	2.9	3.0	2.7	2.7	2.9	2.6
SVM	PERF	0.90	1.07	1.06	0.99	1.11	1.07	1.00	1.19	1.09	1.14	1.40	1.39	1.21	1.38	1.33	1.21	1.47	1.36
	E. EFF	31	26	26	32	27	25	35	28	26	40	34	34	39	34	31	42	35	32
	A. EFF	0.6	0.7	0.7	0.7	0.6	0.6	0.6	0.6	0.6	0.8	0.9	1.0	0.8	0.8	0.8	0.7	0.8	0.8
NAVG	PERF	0.00	0.23	0.24	0.15	0.31	0.27	0.17	0.41	0.31	0.51	0.84	0.80	0.62	0.88	0.77	0.64	1.00	0.81
	E. EFF	0.22	0.05	0.06	0.28	0.10	0.02	0.43	0.15	0.08	0.85	0.58	0.54	0.82	0.59	0.43	1.00	0.64	0.50
	A. EFF	0.03	0.17	0.29	0.16	0.15	0.14	0.08	0.12	0.10	0.67	0.87	0.97	0.73	0.76	0.67	0.61	0.69	0.60

commonly used in DSP: matrix multiplication (MATMUL) and convolution (CONV), which is the most computing-intensive kernel in convolutional neural network (CNN) workloads.

We have implemented different variants of each kernel, using scalar (*float*) and vector ($2 \times float16$, $2 \times bfloat16$) data types. Considering the design of the FPU, there is no significant difference in execution time and energy consumption between *float16* and *bfloat16* vectors; in the following experiments, we report a single value for both configurations.

To exploit the parallelism provided by the transprecision cluster, each variant accepts a parameter representing the

number of cores available in the current configuration. The source code includes a form of parametric parallelism based on the number of available cores and the core id, using the low-overhead HAL interface described in Section 4.

We exploited data parallelism at loop level with static scheduling of the iterations on the available cores. This policy guarantees the maximum balancing with a limited overhead related to the computation of per-core iteration boundaries. Whenever it is feasible, we apply data parallelism to the outer loops of the benchmarks (CONV, FIR, MATMUL). In other cases, data parallelism is applied to single stages of the algorithm, separated by a synchroniza-

tion barrier (DWT, FFT, KMEANS, SVM); except for FFT, these benchmarks are characterized by sequential regions interleaved with parallel loops and executed by a single core.

A common problem of IIR filters working on a single stream is that data dependencies limit the parallelism. To alleviate this limitation, we have adopted a technique based on a block formulation of recursive filters tailored for vector units [45]. The algebraic transformations required by this technique are applied off-line and do not imply any overhead. However, the time complexity of the algorithm is higher than the original one and the size of the vector state (equal to the number of taps) severely limits the exploitability of parallelism. For this reason, the vector variant of this benchmark is the only reported case with alternative configurations achieving the best result for energy efficiency.

Table 3 also reports the FP and memory intensity of the benchmarks for scalar and vector variants. The FP intensity is computed as the ratio between the number of FP instructions and the total number of instructions. Analogously, the memory intensity is the number of load/store instructions over the total number of instructions. These numbers provide a quantitative evaluation of the pressure on the FPU and memory subsystems and are essential to understand the actual FP workload in a real execution scenario.

5.3 Benchmarking

We have performed an extensive benchmarking considering all the benchmark variants and all the configurations of the transprecision cluster. We have measured the performance (Gflop/s), the energy efficiency (Gflop/s/W), and the area efficiency (Gflop/s/mm²) for each benchmark variant and platform configuration: Table 4 and Table 5 report the result of the experiments. The last row of each table reports the normalized average of the measures, computed normalizing the values into values between 0 and 1 (min-max normalization). Computing these metrics allows establishing the configurations that, on average, return the worst and the best performance and energy/area efficiency for the considered benchmarks. Moreover, the tables use a color scale to visually emphasize the worst (light color) and the best (dark color) configurations.

The configuration with 16 cores, private FPUs, and one pipeline stage provides the best performance, with a maximum of 3.37 Gflop/s for scalars and 5.92 Gflop/s for vectors. It is quite intuitive that using the maximum number of cores and FPUs is beneficial for performance. An additional pipeline stage could enable a further increase of the frequency, but this is not the case due to structural critical paths (as discussed in Section 3.3).

The configuration with 16 cores, private FPUs, and zero pipeline stages is the most energy-efficient, with a maximum of 99 Gflop/s/W and 167 Gflop/s/W for vectors. Using the maximum number of cores is never detrimental to performance, mainly thanks to the adoption of aggressive power-saving policies that turn off cores waiting for synchronization events. Moreover, this configuration removes FPU stalls, which are detrimental to energy efficiency.

The configuration with 8 cores and 4 shared FPUs configured with one pipeline stage is the most area-efficient,

with a maximum of 2.0 Gflop/s/mm² for scalars and 3.5 Gflop/s/mm² for vectors. This configuration saves area reducing the number of cores and the sharing factor, but maintaining a single pipeline stage represents the best trade-off with performance.

5.3.1 Parallelization and vectorization

Fig. 6 depicts the speed-ups from the execution of the benchmarks on the 16-cores architectures, combining the benefits deriving from parallelism and vectorization. Each configuration of the transprecision cluster is denoted by the abbreviation n -CL, where n indicates the number of cores. The suffix VECT designates the execution of the vector variant. The baseline to compute the speed-up is the execution on a single core with no vectorization support. The bars show average, maximum, and minimum values of the speed-ups executed on all the architectural configurations.

Focusing on the parallel speed-up, we can notice that the values reported for DWT, IIR, and K-MEANS are modest, reaching a saturation point around 8. These benchmarks have a complex parallel execution flow, requiring several synchronization barriers and regions with sequential execution to ensure the correctness of the results, and this limits the parallelism. However, this effect is not detrimental to energy efficiency, as discussed in Section 5.3. The rest of the kernels (CONV, FFT, FIR, and MATMUL) demonstrate a nearly ideal speed-up.

Vectorization leads to an additional improvement of the speed-up – between $1.3\times$ and $2\times$ – thanks to the beneficial effects described in Section 1. Moreover, the improvement derived from vectorization is higher than the parallel speed-up for some applications. This trend is more evident for FIR, IIR, MATMUL, and KMEANS when passing from 8CL-VECT (8 cores working on vectors) to 16CL (8 cores working on scalars). This effect is due to the different overheads related to parallelization and vectorization. As discussed above, IIR and K-MEANS require several synchronization barriers and regions with sequential execution semantic. Conversely, FIR and MATMUL are amenable to advanced manual vectorization techniques. For instance, the vector variant of MATMUL reaches a near-ideal improvement vectorizing both input matrices. The efficiency is achieved by unrolling the two inner loops, adding shuffle operations to compute the transpose, and using a dot-product intrinsic to accumulate two products. A similar technique is applied to FIR. On the other side, the complex multiplication kernel required by FFT requires 7 cycles for scalar data and 10 cycles for vector data; consequently, the maximum gain from vectorization is $1.43\times$.

5.3.2 Sharing factor

Fig. 7 reports average values of performance, energy efficiency, and area efficiency varying the sharing factor. The left part of the figure references 8-cores configurations, the right part 16-cores ones. The number of pipeline stages has been set to one for all experiments, while the number of FPUs corresponds to sharing factors $1/4$, $1/2$, and $1/1$, respectively.

As a general trend, performance grows when increasing the sharing factor. This increment is more evident passing

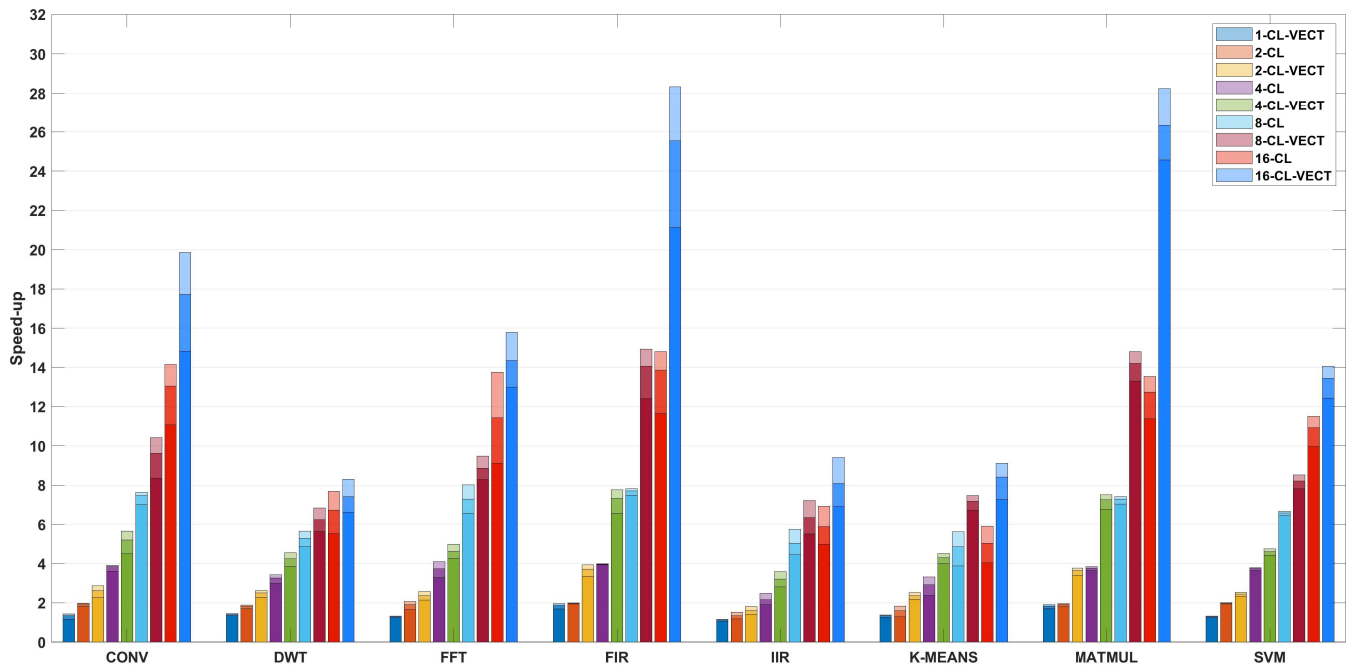


Fig. 6. Speed-ups obtained executing scalar and vector variants on all the platform configurations. Each configuration reports the number of available cores and the support to vectorization. Each bar shows the minimum (dark color), maximum and average (light color) value.

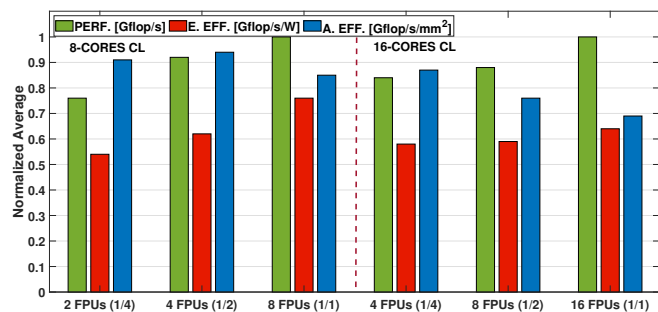


Fig. 7. Performance (PERF.), energy efficiency (E. EFF.), and area efficiency (A. EFF.) fixing one pipeline stage and varying the number of FPUs. The values are the average of the normalized results.

from 1/4 to 1/2 in 8-cores configurations, and passing from 1/2 to 1/1 in the 16-cores configurations.

The energy efficiency increases with the sharing factor. This effect is less evident for 16-cores configurations because the contribution of FPUs to the total energy consumption is proportionally lower. Conversely, the area efficiency increases by reducing the sharing factor from 1/1 to 1/4. This trend is inverted in the transition from 1/4 to 1/2 with 8 cores. This effect is related to the FP intensity of benchmarks, which is always less than one (as expected in real applications). A 1/2 sharing factor can sustain an FP intensity up to 0.5 with no additional stalls. This value is enough for the requirements of most applications, considering that 0.31 is the average FP intensity of the benchmarks in Table 3. On the 16-cores configuration, the number of FPUs to reach the

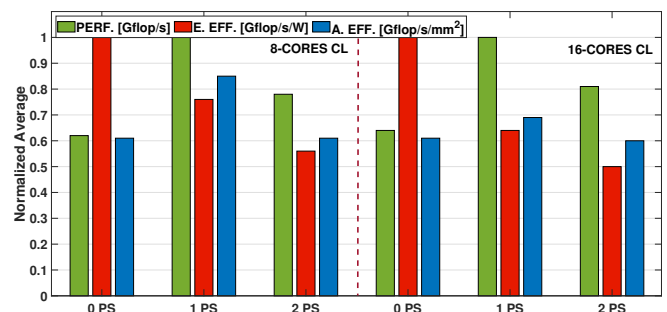


Fig. 8. Performance (PERF.), Energy efficiency (E. EFF.), and area efficiency (A. EFF.) fixing a 1/1 sharing factor and varying the pipeline stages (PS). The values are the average of the normalized results.

same sharing factor is higher, implying a significant increase of the area; in this case, the best area efficiency corresponds to the minimum sharing factor (1/4).

5.3.3 Pipelining

Fig. 8 shows average values of performance, energy efficiency, and area efficiency varying the number of pipeline stages. The support for pipelining improves performance since this technique allows for increasing the operating frequency of the transprecision cluster. Conversely, performance degrades with two pipeline stages. Even if we can increase the operating frequency, we observe an increment in the number of cycles due to the contentions on the write ports of the register file. Configuring the FPU for two pipeline stages, a write-back stall may happen when a

TABLE 6
Comparison with state-of-the-art architectures in high-performance and low-power embedded domain.

	Ara [27]	Hwacha [28]	Snitch [42]	Ariane [41]	NTX [41]	Xavier*	STM32H7†	Mr.Wolf [2]	This work Best perf. (16c16f1p) Best en. eff. (16c16f0p) Best area eff. (8c4f1p)
Domain	High-perf.	High-perf.	High-perf.	High-perf.	High-perf.	Embedded	Embedded	Embedded	Embedded
Technology	GF 22FDX	45nm SOI	GF 22FDX	GF 22FDX	GF 22FDX	TSMC 12FFN	40nm CMOS	40nm CMOS	GF 22FDX
Voltage (V)	0.80 ²	0.80 ¹	0.80 ²	0.80 ¹	0.80 ¹	0.75 ¹	1.80 / 1.80 ¹	1.10 ¹	0.80 / 0.65 / 0.80 ³
Frequency (GHz)	1.04	0.55	1.06	0.92	1.55	1.38	0.20 / 0.48	0.45	0.37 / 0.30 / 0.43
Area (mm²)	2.14	3.00	0.89	0.39	0.56	11.03	–	10.00	2.10 / 1.80 / 0.97
Performance (Gflop/s)	64.80	3.44	14.38	2.04	18.27	153.00	0.03 / 0.07	1.00	2.86 / 2.30 / 1.74
Energy eff. (Gflop/s/W)	81.60	25.00	103.84	33.02	110.05	52.39	0.44 / 0.33	4.50	26.00 / 81.00 / 23.40
Area eff. (Gflop/s/mm²)	30.34	1.14	25.83	5.23	32.63	13.84	–	1.70	1.50 / 0.60 / 1.78
FP formats	float float16 bfloat16 minifloat	double float	double float	float float16 bfloat16 minifloat	float [‡]	float float16	float	float	float float16 bfloat16
Programming interface	ISA extension	ISA extension	ISA extension	ISA extension	Memory-mapped configuration	Base ISA	Base ISA	Base ISA	ISA extension
Execution model	SIMD vector unit (accelerator)	SIMT vector-thread unit (accelerator)	Loop-buffers for tensor streaming (accelerator)	SIMD processor	Loop-buffers for tensor streaming (accelerator)	SIMT vector-thread unit (accelerator)	Processor	Multi-core processor	Multi-core processor
Compiler support	Yes	Yes (OpenCL)	Partial (inline ASM)	Yes	No	Yes (CUDA)	Yes	Yes	Yes

* Numbers extracted from [42]. † Measurements taken on a NUCLEOH743ZI development board executing a 128×128. matrix multiplication.

¹ Silicon measurements. ² Post-layout simulation using *typical* frequency. ³ Post-layout simulation using *worst-case* frequency.

[‡] Higher internal accumulation precision with float results.

load/store post-increment operation or an integer operation arrives right after an FP operation. For instance, when we have the valid signal for the FP operation in the first cycle, and then a load/store post-increment request in the second clock cycle before storing the FP results, the FPU must wait until the other instructions end, resulting in a stall for the use of the port. There are no contentions with no pipeline stages because there is a dedicated port for the FPU.

In all cases, energy efficiency decreases by incrementing the number of pipeline stages since the design makes the logic more complicated. Finally, area efficiency follows a trend very similar to performance. The area required to enable one-stage pipelining leads to a considerable benefit while adding additional area for a second stage is not so convenient. This trend is less evident on 16-cores configurations since the impact of pipeline logic is less significant.

6 COMPARISON WITH THE SOA

Table 6 depicts a comparison with SoA platforms with FP support in high-performance and embedded domains. The number of FP operations has been measured by executing a single-precision matrix multiplication on all the platforms. We have considered three configurations of the transprecision cluster (reported in the TP column), corresponding to the best performance, the best energy efficiency, and the best area efficiency. In the domain of low-power embedded systems, our solution outperforms a single-core Cortex core and the Mr.Wolf multi-core cluster in all metrics. The Tegra Xavier SoC contains eight streaming multiprocessors (SMs) composed of four execution units. Each execution unit includes 16 single-precision FPUs sharing a register file and an instruction cache. The transprecision cluster is 53%

better than an SM in terms of energy efficiency. As regards performance, a single execution unit is 13× faster.

As expected, the absolute performance and area efficiency of platforms in the high-performance domain is higher than the transprecision cluster due to the higher operating frequencies. In our design, we consider the worst-case corner for the computation of the operating frequency, while the other solutions report silicon results or typical corners, which is somehow penalizing for us. Nevertheless, our solution outperforms an Ariane and is comparable with a Hwacha vector processor. The energy efficiency of the transprecision cluster is comparable with Snitch, NTX, and Ara, despite these architectures are heavily specialized for FP intensive computations. This outcome is due to three main factors. First, operating at low voltage in near-threshold operation makes the transprecision cluster very power efficient. Second, the best solution is not to adopt pipelining, so it does not pay the energetic overhead of pipeline logic. Third, the support to FMA operations increments by 2× the number of operations performed per cycle and is highly beneficial.

Finally, compared to most energy-efficient solutions in Table 6, the proposed cluster provides full compiler support and flexibility typical of high-level parallel programming models such as OpenMP, not requiring programmers to use low-level accelerator-centric interfaces such as OpenCL or memory-mapped APIs, or even lower-level abstractions (e.g., inline assembly). This support is a key requirement for the wide adoption of these solutions for near-sensor computing.

7 CONCLUSION

In this paper, we have described the design of a transprecision cluster for near-sensors computing, providing a full specification of its main components and a software ecosystem to execute real applications. We have performed a design space exploration on an FPGA emulator varying the number of cores, the number of FPU's, and the pipeline stages. A set of experiments on near-sensor applications and an analysis on post P&R models have allowed us to identify the most efficient configurations.

Our experimental results show that configurations with 16 cores and private FPU's are the best solution in terms of performance and energy efficiency, while a cluster with 8 cores and 4 shared FPU's remains the best solution for area efficiency. Moreover, these results highlight two important outcomes. First, energy efficiency is not affected by the effectiveness of parallelization techniques since the platform provides effective power-saving policies to turn off cores that are not active. Second, the trend for energy efficiency is different from area efficiency; in the design space that we are considering, these metrics are related but do not scale with a fixed rate. To conclude, one pipeline stage is the solution providing the best compromise in most configurations; no pipelining can be beneficial only when energy saving is a high-priority constraint. These outcomes provide a useful insight to system designers and engineers, and the guidelines derived by this exploration can steer the design of future near-sensor computing platforms in a wide range of application domains.

REFERENCES

- [1] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini, "A transprecision floating-point platform for ultra-low power computing," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1051–1056.
- [2] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "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.
- [3] B. Barrois and O. Sentieys, "Customizing fixed-point and floating-point arithmetic case study in k-means clustering," in *2017 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2017, pp. 1–6.
- [4] A. Volkova, T. Hilaire, and C. Lauter, "Arithmetic Approaches for Rigorous Design of Reliable Fixed-Point LTI Filters," *IEEE Transactions on Computers*, vol. 69, no. 4, pp. 489–504, 2020.
- [5] A. C. I. Malossi, M. Schaffner, A. Molnos, L. Gammaitoni, G. Tagliavini, A. Emerson, A. Tomás, D. S. Nikolopoulos, E. Flamand, and N. Wehn, "The transprecision computing paradigm: Concept, design, and applications," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1105–1110.
- [6] D. Zuras, M. Cowlshaw, A. Aiken, M. Applegate, D. Bailey, S. Bass, D. Bhandarkar, M. Bhat, D. Bindel, S. Boldo *et al.*, "IEEE standard for floating-point arithmetic," *IEEE Std*, vol. 754, no. 2008, pp. 1–70, 2008.
- [7] M. Zanghieri, S. Benatti, A. Burrello, V. Kartsch, F. Conti, and L. Benini, "Robust real-time embedded emg recognition framework using temporal convolutional networks on a multicore iot processor," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 2, pp. 244–256, 2020.
- [8] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.
- [9] N. Burgess, J. Milanovic, N. Stephens, K. Monachopoulos, and D. Mansell, "Bfloat16 processing for neural networks," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2019, pp. 88–91.
- [10] G. Tagliavini, A. Marongiu, and L. Benini, "FlexFloat: A software library for transprecision computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.
- [11] A. A. Wahba and H. A. Fahmy, "Area efficient and fast combined binary/decimal floating point fused multiply add unit," *IEEE Transactions on Computers*, vol. 66, no. 2, pp. 226–239, 2016.
- [12] M. Gautschi, M. Schaffner, F. K. Grkaynak, and L. Benini, "An Extended Shared Logarithmic Unit for Nonlinear Function Kernel Acceleration in a 65-nm CMOS Multicore Cluster," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 98–112, 2017.
- [13] J. L. Gustafson, *The End of Error: Unum Computing*. CRC Press, 2017.
- [14] F. Glaser, S. Mach, A. Rahimi, F. K. Grkaynak, Q. Huang, and L. Benini, "An 826 MOPS, 210uW/MHz Unum ALU in 65 nm," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.
- [15] A. Bocco, Y. Durand, and F. De Dinechin, "SMURF: Scalar Multiple-precision Unum Risc-V Floating-point Accelerator for Scientific Computing," in *Proceedings of the Conference for Next Generation Arithmetic 2019*, 2019, pp. 1–8.
- [16] J. L. Gustafson and I. T. Yonemoto, "Beating Floating Point at its Own Game: Posit Arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, pp. 71–86, 2017.
- [17] H. Kaul, M. Anders, S. Mathew, S. Hsu, A. Agarwal, F. Sheikh, R. Krishnamurthy, and S. Borkar, "A 1.45GHz 52-to-162GFLOPS/W variable-precision floating-point fused multiply-add unit with certainty tracking in 32nm CMOS," in *2012 IEEE International Solid-State Circuits Conference*. IEEE, 2012, pp. 182–184.
- [18] A. Nannarelli, "Tunable Floating-Point Adder," *IEEE Transactions on Computers*, vol. 68, no. 10, pp. 1553–1560, 2019.
- [19] M. K. Jaiswal, B. S. C. Varma, H. K. . So, M. Balakrishnan, K. Paul, and R. C. C. Cheung, "Configurable Architectures for Multi-Mode Floating Point Adders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 8, pp. 2079–2090, 2015.
- [20] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.
- [21] H. Zhang, D. Chen, and S. Ko, "Efficient Multiple-Precision Floating-Point Fused Multiply-Add with Mixed-Precision Support," *IEEE Transactions on Computers*, vol. 68, no. 7, pp. 1035–1048, 2019.
- [22] S. Mach, F. Schuiki, F. Zaruba, and L. Benini, "FPnew: An Open-Source Multi-Format Floating-Point Unit Architecture for Energy-Proportional Transprecision Computing," 2020.
- [23] J. D. Bruguera, "Low Latency Floating-Point Division and Square Root Unit," *IEEE Transactions on Computers*, vol. 69, no. 2, pp. 274–287, 2019.
- [24] R. M. Russell, "The CRAY-1 computer system," *Communications of the ACM*, vol. 21, no. 1, pp. 63–72, 1978.
- [25] N. Stephens, S. Biles, M. Boettcher, J. Eapen, M. Eyole, G. Gabrielli, M. Horsnell, G. Magklis, A. Martinez, N. Premillieu *et al.*, "The ARM scalable vector extension," *IEEE Micro*, vol. 37, no. 2, pp. 26–39, 2017.
- [26] T. Yoshida, "Fujitsu high performance CPU for the Post-K Computer," in *Hot Chips*, vol. 30, 2018.
- [27] M. Cavalcante, F. Schuiki, F. Zaruba, M. Schaffner, and L. Benini, "Ara: A 1-GHz+ Scalable and Energy-Efficient RISC-V Vector Processor With Multiprecision Floating-Point Support in 22-nm FD-SOI," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 530–543, 2019.
- [28] Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, V. Stojanović, and K. Asanović, "A 45nm 1.3 GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators," in *ESS-CIRC 2014–40th European Solid State Circuits Conference (ESSCIRC)*. IEEE, 2014, pp. 199–202.
- [29] D. H. Bailey, H. Yozo, X. S. Li, and B. Thompson, "ARPREC: An arbitrary precision computation package," *Lawrence Berkeley National Laboratory*, 2002.
- [30] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélicier, and P. Zimmermann, "MPFR: A multiple-precision binary floating-point library with correct rounding," *ACM Transactions on Mathematical Software (TOMS)*, vol. 33, no. 2, p. 13, 2007.
- [31] F. Johansson, "Arb: efficient arbitrary-precision midpoint-radius interval arithmetic," *IEEE Transactions on Computers*, vol. 66, no. 8, pp. 1281–1292, 2017.

- [32] V. Lefèvre, "Correctly rounded arbitrary-precision floating-point summation," *IEEE Transactions on Computers*, vol. 66, no. 12, pp. 2111–2124, 2017.
- [33] A. Anderson, S. Muralidharan, and D. Gregg, "Efficient Multibyte Floating Point Data Formats Using Vectorization," *IEEE Transactions on Computers*, vol. 66, no. 12, pp. 2081–2096, 2017.
- [34] M. Brandalero, L. Carro, A. C. S. Beck Filho, and M. Shafique, "Multi-Target Adaptive Reconfigurable Acceleration for Low-Power IoT Processing," *IEEE Transactions on Computers*, 2020.
- [35] R. Prasad, S. Das, K. Martin, G. Tagliavini, P. Coussy, L. Benini, and D. Rossi, "TRANSPiRE: An energy-efficient TRANSprecision floating-point Programmable archItectuRE," in *Design, Automation and Test in Europe Conference (DATE)*, 2020.
- [36] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.
- [37] N.-M. Ho and W.-F. Wong, "Exploiting half precision arithmetic in Nvidia GPUs," in *High Performance Extreme Computing Conference (HPEC), 2017 IEEE*. IEEE, 2017, pp. 1–7.
- [38] S. Eliuk, C. Upright, and A. Skjellum, "dMath: A Scalable Linear Algebra and Math Library for Heterogeneous GP-GPU Architectures," *arXiv:1604.01416 [cs.EU]*, 2016. [Online]. Available: <http://arxiv.org/abs/1604.01416>
- [39] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "Neurostream: Scalable and energy efficient deep learning with smart memory cubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 2, pp. 420–434, 2017.
- [40] F. Schuiki, M. Schaffner, F. K. Gürkaynak, and L. Benini, "A scalable near-memory architecture for training deep neural networks on large in-memory datasets," *IEEE Transactions on Computers*, vol. 68, no. 4, pp. 484–497, 2018.
- [41] F. Zaruba, F. Schuiki, S. Mach, and L. Benini, "The Floating Point Trinity: A Multi-modal Approach to Extreme Energy-Efficiency and Performance," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2019, pp. 767–770.
- [42] F. Zaruba, F. Schuiki, T. Hoefler, and L. Benini, "Snitch: A 10 kGE Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads," *arXiv preprint cs.AR/2002.10143*, 2020.
- [43] F. Glaser, G. Haugou, D. Rossi, Q. Huang, and L. Benini, "Hardware-Accelerated Energy-Efficient Synchronization and Communication for Ultra-Low-Power Tightly Coupled Clusters," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. IEEE, 2019, pp. 552–557.
- [44] F. Montagna, M. Buiatti, S. Benatti, D. Rossi, E. Farella, and L. Benini, "A machine learning approach for automated wide-range frequency tagging analysis in embedded neuromonitoring systems," *Methods*, vol. 129, pp. 96–107, 2017.
- [45] J. Robelly, G. Cichon, H. Seidel, and G. Fettweis, "Implementation of recursive digital filters into vector SIMD DSP architectures," in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5. IEEE, 2004, pp. V–165.



Fabio Montagna received the Ph.D. degree in Electrical, Electronic and Information Engineering from the University of Bologna, Bologna, Italy, in 2020. He is currently working as Research Fellow at DISI, University of Bologna, Bologna, Italy. His main research topic is energy-efficient parallel architectures for ultra-low power biosignal processing. His research interests include embedded wearable and implantable systems, parallel computing, signal processing, and machine learning.



Stefan Mach received his B.Sc. and M.Sc. degree from the Swiss Federal Institute of Technology Zurich (ETHZ), Switzerland, where he is currently pursuing a Ph.D. degree. Since 2017, he has been a research assistant with the Integrated Systems Laboratory at ETHZ. His research interests include transprecision computing, computer arithmetics, and energy-efficient processor architectures.



Electronic Designer and R&D Engineer of electromedical devices.

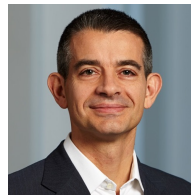


programmable embedded architectures.



Angelo Garofalo received the B.Sc and M.Sc. degree in electronic engineering from the University of Bologna, Bologna, Italy, in 2016 and 2018 respectively. He is currently working toward his Ph.D. degree at DEI, University of Bologna, Bologna, Italy. His main research topic is Hardware-Software design of ultra-low power multiprocessor systems on chip. His research interests include Quantized Neural Networks, Hardware efficient Machine Learning, transprecision computing, and energy-efficient fully-

Gianmarco Ottavi is a researcher at University of Bologna (Italy) in the department of Electrical, Electronic and Information Engineering (DEI). His current research topics are on developing Ultra Low Power embedded systems based on RISC-V Instruction Set Architecture.



Luca Benini holds the chair of digital Circuits and systems at ETHZ and is Full Professor at the Università di Bologna. Dr. Benini's research interests are in energy-efficient computing systems design, from embedded to high-performance. He has published more than 1000 peer-reviewed papers and five books. He is a Fellow of the ACM and a member of the Accademia Europaea. He is the recipient of the 2016 IEEE CAS Mac Van Valkenburg Award and the 2020 EDAA Achievement Award.



O. Pederson Best Paper Award 2018, 2020 IEEE TCAS Darlington Best Paper Award, 2020 IEEE TVLSI Prize Paper Award.

Davide Rossi received the Ph.D. degree from the University of Bologna, Bologna, Italy, in 2012. He has been a Post-Doctoral Researcher with the Department of Electrical, Electronic and Information Engineering Guglielmo Marconi, University of Bologna, since 2015, where he is currently an Assistant Professor. His research interests focus on energy-efficient digital architectures. In this field, he has published more than 100 papers in international peer-reviewed conferences and journals. He is recipient of Donald



emerging computing architectures.

Giuseppe Tagliavini received the Ph.D. degree in electronic engineering from the University of Bologna, Bologna, Italy, in 2017. He is currently an Assistant Professor with the Department of Computer Science and Engineering (DISI) at the University of Bologna. He has co-authored over 30 papers in international conferences and journals. His research interests include parallel programming models for embedded systems, runtime optimization for multicore and many-core accelerators, and design of software stacks for