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

(Article begins on next page)

11 August 2024

A Low-Power 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 low-power (1-20 mW) 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 low-power 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 solution – based on the open-source RISC-V architecture – combines parallelization and sub-word vectorization with a dedicated interconnect design capable of sharing floating-point units (FPUs) among the cores. On top of this architecture, we provide a full-fledged software stack support, including a parallel low-level runtime, a compilation toolchain, and a high-level programming model, with the aim to support the development of end-to-end applications. We performed an exhaustive exploration of the design space of the transprecision cluster on a cycle-accurate FPGA emulator, varying the number of cores and FPUs to maximize performance. Orthogonally, we performed a vertical exploration to identify the most efficient solutions in terms of non-functional requirements (operating frequency, power, and area). We conducted an experimental assessment 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. 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. Finally, a real-life use case demonstrates the effectiveness of our approach in fulfilling accuracy constraints.

Index Terms—RISC-V, transprecision, parallel computing, sub-word vectorization, FPU interconnect, near-sensor computing

◆

1 INTRODUCTION

THE pervasive adoption of edge computing is increasing the computational demand for algorithms targeted on low-power embedded devices operating in the mW range. Besides the aggressive optimization strategies adopted on the algorithmic side, there is a great effort to find the best trade-off between architectural features and computational capabilities [1]. 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 [2]. Nevertheless, many applications require high precision results characterized by a wide dynamic range (e.g., the accumulation stage of support vectors or feed-

---

- 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

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. This methodology implies 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 [3]). 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 [2]. 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, the 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 when the core executes fixed-point operations, resulting in a further reduction energy efficiency reduction.

In this scenario, *transprecision computing* [4] is emerging

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

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

[1] Number of bits.   [2] Decimal digits.

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 [5]. The IEEE 754-2008 standard [6] describes five FP formats, including the 16-bit half-precision format (*float16*) and the 32-bit 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 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) or Convolutional Neural Networks (CNN), tolerate lower precision arithmetic without losing their accuracy [7]. Adopting the smaller format that satisfies the application accuracy requirements paves the way for substantial improvements in performance and energy consumption. For instance, Quantized Neural Networks (QNNs) encode weights or activations into 8-bit or smaller data types with a negligible accuracy loss – around 2% between 8-bit and float models of a MobileNet SSD topology [8]. Nevertheless, programmers need full software support in the compilation toolchain and also a consolidated methodology for tuning the precision of FP variables [9]. Table 1 provides an overview of the discussed formats.

A major optimization enabled by FP bitwidth reduction relies on applying the single-instruction-multiple-data (SIMD) approach on multiple sub-word elements simultaneously. These data types are known as *packed-SIMD vectors*. As a clear benefit, SIMD operations act on multiple data elements of the same size and type simultaneously, offering a theoretical $2\times$ speed-up for 16-bit data. Moreover, vectorization of 16-bit 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 total number of FPUs and their sharing among the cluster cores require careful evaluation since these aspects directly impact area and energy efficiency. For instance, having a dedicated FPU for each core 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 [10]. Thus, reducing the number of FPUs and adopting a sharing policy among the cores can be beneficial to improve overall 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 potential deterioration of performance if pipeline latency cannot be completely hidden. In this complex 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 of 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 low-power (1-20 mW) near-sensor computing and its complete software ecosystem. The main contributions of our work are:

- The architectural design of the transprecision cluster. A dedicated design for the FPU interconnect enables multiple policies for FPU sharing, with the possibility to guarantee different trade-offs.
- A complete software infrastructure that includes a low-level runtime environment to enable efficient parallel programming, an extended compilation toolchain to support transprecision features, and a high-level programming model (OpenMP) highly optimized to exploit the architectural features.
- 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.
- A vertical exploration to identify the most efficient solutions optimizing non-functional requirements (i.e., operating frequency, 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. This exploration provides guidelines to instantiate an optimal proper cluster configuration depending on the target application domain and expected performance.
- A comparison with 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.
- The description of a use case that demonstrates the effective adoption of the transprecision cluster and

its software infrastructure in a real-life application. This contribution shows how multiple formats can be used and the consequences of mixed-precision computations on application accuracy.

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 the configuration with 8 cores and 4 shared FPUs configured with one pipeline stage is the most area-efficient (3.5 Gflop/s/mm^2). Ultimately, 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

This section provides an overview of the current SoA. First, we consider the main alternatives available for platform and component design. Then, we explore the role of packed-SIMD and vector units in embedded platforms, as they are central to our discussion. We also discuss software-based approaches since they represent a platform-agnostic alternative to our solution. Finally, we provide an overview of both low-power and high-end embedded systems that provide FP support because these architectures provide us a baseline to compare our results.

2.1 Non-IEEE floating-point formats

In recent years, researchers have started to explore custom formats that are alternative to the IEEE standard ones and their closer derivatives (e.g., bfloat16) [7]. Coleman et al. [11] propose an FPU based on the logarithmic number system (LNS) that is almost $2.5\times$ faster than standard FP units in non-linear processing kernels. Alternatively, Universal numbers (*UNUMs*) adopt a variable-width representation based on interval arithmetic to guarantee that the result contains the exact solution [12]. The variable width provided by UNUM enables to scale up precision in scientific computing applications [13], but current hardware implementations are 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* [14], proposes a solution to the hardware overhead issue. Tiwari et al. [15] introduce a posit arithmetic unit supporting multiply-and-add, division, and square root operations. The synthesis of the IEEE-754 and posit units on a 65 nm technology node shows that the area of the posit design is about $1.6\times$ higher. At the same time, the number of cycles required for execution is almost equivalent.

The discussed solutions may represent a viable alternative to reduce execution time in our target domain. However, we have not included these hardware components in the transprecision cluster design for two main reasons. First, current hardware implementations are characterized by high power and area overheads compared to actual performance benefits [16]. Second, their adoption imposes a significant effort by programmers since these approaches are not natively supported as language data types.

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-bit 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 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, and 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 Packet-SIMD support and vector units

Intel initially introduced Packed-SIMD extensions for FP computations with the MMX and SSE ISAs. ARM later introduced the NEON extension, which supports up to 128-bit single-precision FP operations, which has been evaluated as a better solution than the Intel counterpart in low-power embedded platforms [24]. In this context, heterogeneous architectures have also provided support for packed-SIMD instructions featuring a DSP accelerator, such as Texas Instruments Keystone II [25]. Considering the impact of packed-SIMD instructions on the processing capabilities of DSP platforms, the RISC-V consortium is working on a dedicated ISA extension [26].

Variable-length vector units have been initially introduced on the CRAY-1 architecture [27], and today they are a well-established solution in high-end computer systems. The ARM Scalable Vector Extension (SVE) [28] 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 [29]. 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 extension [30] 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). Hwacha [31] is the first embodiment of this provisional standard. It is based on the vector-fetch paradigm and is composed of an array of single-issue, in-order RISC-V Rocket cores [31]. In general, the area and power consumption of these solutions are too high for low-power, MCU-class processing systems. This 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 [32] and GNU MPFR [33] 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 [34] 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 a 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 [35] 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: Time and energy efficiency are negatively affected by at least an order of magnitude compared with solutions based on dedicated hardware.

Anderson et al. [36] 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

FPGAs are among the most promising embedded architectures for signal processing. The recent development of heterogeneous SoCs, such as the Xilinx Zynq family, has enabled a high level of flexibility to build heterogeneous acceleration systems. However, the DSP-capable hardware of these devices requires a power envelope in the order of Watts. Modulating numerical precision of floating-point operation is an efficient method to enable fast, accurate computation for demanding algorithms in near-sensor processing. Nevertheless, the design of FPUs on a low-power FPGA, such as Lattice senseAI [37], is hampered by the reduced LUTs capabilities of these devices as well as the lack of DSP blocks with transprecision support.

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 [38] 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 [39] is a CGRA architecture with FP support. The authors state an improvement of around $10\times$ in performance and energy efficiency compared with a RISC-V core extended with packed-SIMD vectorization. These benefits are limited to specific algorithms since 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. For example, Prasad et al. [39] present the implementation of a Principal Component Analysis (PCA) algorithm. Although Transpire delivers

remarkable efficiency, the performance of a tightly-coupled cluster of processors scales much better with the number of cores. Moreover, the work presented in [39] does not consider the performance and energy reconfiguration overhead needed to swap the different kernels of the PCA. The results presented in Section 6 show that exploiting clusters of software programmable processors leads to better performance in a much more software-friendly programming environment.

Mr.Wolf [1] is a multi-core programmable processor implemented in CMOS 40nm technology. The platform includes a tiny (12 Kgates) RISC-V core accelerated by a powerful 8-cores cluster of RI5CY cores 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.

Helium [40] 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-M55 [41] 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 are naturally robust to errors. These features of modern GPUs have also been exploited in other application domains, such as linear algebra [42], demonstrating significant benefits for performance and efficiency. NVidia Pascal has been the first GPU with native support for 16-bit FP formats. The Volta microarchitecture further extended support to reduced precision types featuring mixed-precision multiply-and-add instructions; the new Ampere microarchitecture improves performance by 2.5×, increasing to 5× with the support for sparse computations. However, GP-GPUs are optimized for throughput-oriented application domains, while our platform design prioritizes the energy and area constraints of near-sensor applications.

Other research works target neural network training and FP-intensive workloads by leveraging specialized architectures. NTX is a memory-mapped streaming co-processor targeting inference and training of deep neural networks in high-performance computing systems [43]. This design removes the register-file bottleneck of SIMD architectures accessing the memory exploiting programmable hardware
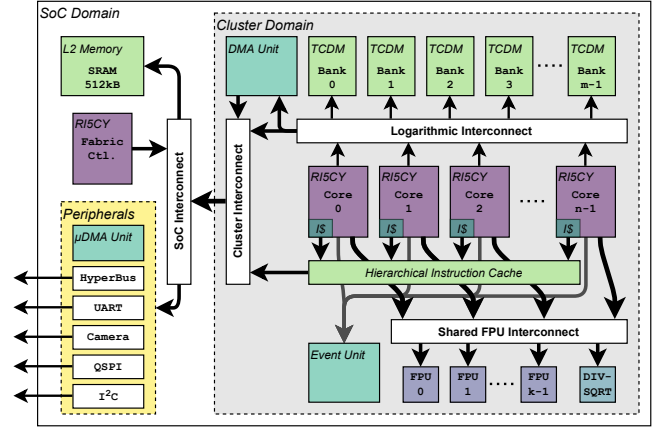

Fig. 1. Top-level view of the proposed transprecision cluster.

loops and address generators, and enabling execution efficiency close to one MAC per cycle. NTX improves energy efficiency by 2.7× over GPGPUs, with 4.4× less silicon area, delivering 1.2 TFLOP/s.

In [44], the memory-mapped control has been replaced by a tiny general-purpose processor meant to drive double-precision FPUs, improving the 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× multi-core speed-up and a 3.5× 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 valuable insight and is discussed in Section 6.

3 ARCHITECTURE AND IMPLEMENTATION

This section illustrates the architecture of the transprecision cluster, with a specific focus on the FPU design choices. It also presents the results deriving from the physical implementation and the related design space exploration.

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 RISC-V cores called *RI5CY*. The cluster is integrated into an SoC featuring a standard set of peripherals such as a JTAG interface used to pre-load program and data into an L2 scratchpad memory accessed by the cluster through an AXI bus. Fig.1 shows the top-level design of the transprecision cluster.

RI5CY 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. 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*. This design choice is a standard solution adopted in the low-power computing domain to avoid area and energy overheads associated with data caches [45]. From the software perspective, data coherency is enforced by algorithms exploiting inter-core synchronization primitives. 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. The event unit also controls the clock gating of the cores within the cluster. The event unit clock-gates the cores waiting on a synchronization barrier until all the other cores reach the barrier; at this point, all cores can resume the program flow.

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 (4 × 128 kB banks), the TCDM is 64 kB for the 8-core configurations (16 × 4 kB banks) and 128 kB for the 16-core ones (32 × 4 kB banks). 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 RI5CY 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 for smaller-than-32-bits data types, and the insertion of any number of pipeline stages. The last parameter is a key configuration knob for the design space exploration discussed in this paper. 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
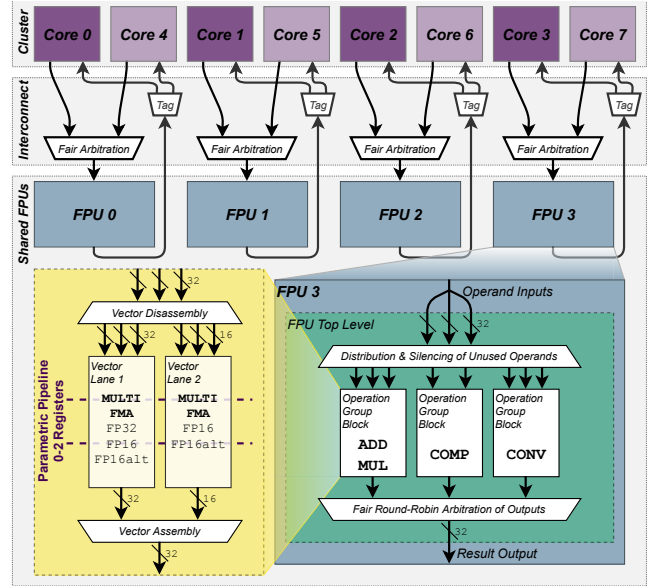

Fig. 2. FPU sharing for the 8-core, 4-FPU configuration.

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 interconnect interface 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

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

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.
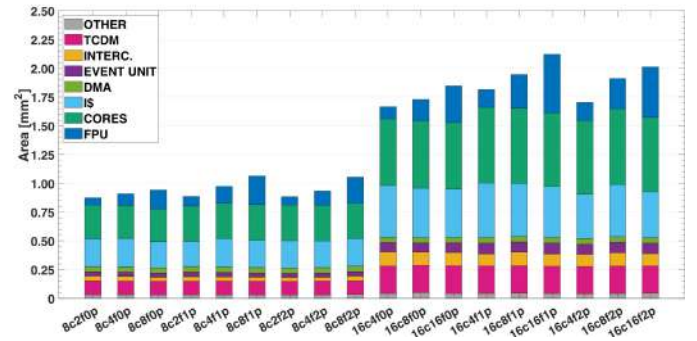

Fig. 4. Total area of all the configurations in the design space of the transprecision cluster.

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 these design choices 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 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

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
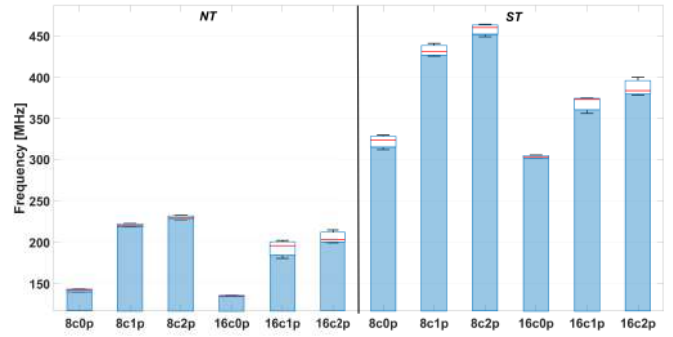
all permutations of operating conditions for signoff: fast and slow process transistors, $125°C$ and $-40°C$ temperatures, $\pm10\%$ of the voltage supply, as well as optimistic and pessimistic parasitics. Power analysis has been performed with Synopsys PrimeTime 2019.12 using the nominal corners at 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. This frequency supports a power consumption in the range of 2-4 mW. In Fig. 3, we report the minimum, maximum, and median values of the frequencies obtained varying the number of cores (*c*) and pipeline stages (*p*). This analysis does not take into account the total number of FPUs (*f*) since this parameter does not affect the outcomes. 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.
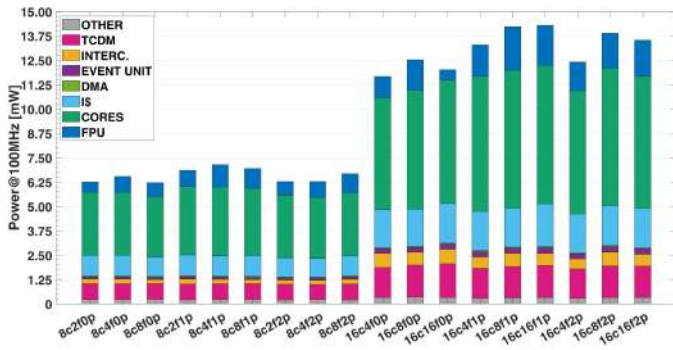
Fig. 5. Total power consumption (at 100 MHz) of all the configurations in the design space of the transprecision cluster.

When moving to single-stage pipeline solutions, we note a 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 in 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 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 scaling 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 I\$.

4 SOFTWARE INFRASTRUCTURE

The exploitation of the transprecision cluster in end-to-end applications requires a full software stack. For this purpose, we designed complete support for scalar and vector data types for the C/C++ languages, advanced optimizations to support ISA extensions in the compiler toolchain, and a parallel programming model to raise the level of abstraction. All these tools are available as open-source projects on the PULP code repository [46].

4.1 Scalar types

To enable the adoption of the new formats, we have extended the GCC front-end for C/C++ programs with new data types, namely *float16*, *bfloat16*. In the usual cross-compilation circumstance, the FP constants in the programs must be represented in the target format to avoid range and precision errors. This requirement determines that the cross compiler cannot use the host FP arithmetic; for this purpose, GCC includes an FP emulator to deal with constants in optimization passes (e.g., constant propagation). We extended this emulator to manage the new formats.

Finally, we extended the type system to include rules for automatic type promotion in mixed-typed expressions involving the new FP types. In accordance with the standard rules, promotion is based on the bit width. Consequently, a float16/bfloat16 operand is converted to float if the expression includes a float operand. As an additional rule, bfloat16 operands are converted to float16 in expressions that include only 16-bit types to avoid indeterminacy.

4.2 Vector types

Adopting the GCC approach for vector types, programmers can specify 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 the performance and energy efficiency of transprecision computing techniques. A set of compiler intrinsics provides access to cast-and-pack operations.

The automatic vectorization pass of GCC operates on the middle-end intermediate representation [47]. 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).

4.3 Compiler back-end

In addition to the middle-end passes described in the previous subsection, we further extended the compiler at the

back-end level to support a parametric number of FPU pipeline stages. This parameter substantially impacts 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 latencies and reservation delays for the functional units involved in FP operations.

Finally, we specified a set of platform-specific parameters for the instruction scheduling algorithm. The GCC 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.

4.4 Hardware Abstraction layer (HAL)

The architectural template of the transprecision cluster promotes a *Single-Program Multiple-Data* (SPMD) parallel paradigm. This approach is supported by a Hardware Abstraction Layer (HAL), which allows access to the platform features with minimal overhead. 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. With this approach, 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.

4.5 High-level parallel programming model

As a high-level approach to code parallelization, we support OpenMP [48], a parallel programming model based on directives that are translated by the compiler front-end into functions calls towards a runtime environment. OpenMP relies on the *fork/join* programming paradigm. A program based on this conceptual framework executes sequentially on a single thread and creates additional threads (*fork*) to exploit parallelism in annotated code regions. After the completion of the parallel workload, the program encounters a synchronization point (*join*), after which it continues sequentially.

We optimized the OpenMP runtime to reduce three main sources of overheads that are relevant in our target domain. First, we reduced the overhead associated with parallel regions by removing support for some recent features. Our implementation is compliant with the OpenMP 3.0 specification, except for the *task* directive. In real applications that we considered, this support is sufficient for the common requirements of this class of devices. Second, we re-designed the synchronization primitives, which depend on the POSIX API in the GCC reference implementation, intending to

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

exploit the benefits of the Event Unit (introduced in Section 3.1). This aspect is crucial to reduce the overhead of barriers and critical sections, which can become relevant if compared with the optimized HAL primitives described in Section 4.4. Third, we modified the directive lowering performed by the GCC toolchain. Instead of calling runtime functions and outlining the user code, loops with static scheduling policy and implicit barriers are directly inlined. This last point is a key enabler for the adoption of OpenMP: A high-level model can provide a more intuitive interface than HAL primitives, but it comes at the cost of higher overhead when parallel workloads are fine-grained.

5 EXPERIMENTAL RESULTS

This section describes the experimental setup, the benchmark suite, and the main outcomes of the experiments.

5.1 Experimental Set-up

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

TABLE 4

Performance [$Gflop/s$], energy efficiency [$Gflop/s/W$], and area efficiency [$Gflop/s/mm^2$] executing the benchmarks on the 8- and 16-cores configurations. Performance and area efficiency are computed at 0.8 V, energy efficiency at 0.65 V. Each cell reports the best configuration name and the corresponding value for the metric. The last line reports normalized average values.

	8 CORES						16 CORES					
	Scalar			Vector			Scalar			Vector		
	Perf.	**En. Eff.**	**Area Eff.**	**Perf.**	**En. Eff.**	**Area Eff.**	**Perf.**	**En. Eff.**	**Area Eff.**	**Perf.**	**En. Eff.**	**Area Eff.**
CONV	8c8f1p 2.04	8c8f0p 91	8c4f2p 2.1	8c8f1p 2.98	8c8f0p 139	8c4f1p 3.0	**16c16f1p 3.37**	16c16f0p 94	16c4f2p 2.0	16c16f1p 4.78	16c16f0p 140	16c4f1p 2.7
DWT	8c8f1p 0.95	8c8f0p 45	8c4f2p 0.9	8c8f1p 1.21	8c8f0p 55	8c4f2p 1.3	16c16f1p 1.06	16c16f0p 31	16c4f2p 0.6	16c16f1p 1.11	16c16f0p 33	16c4f2p 0.8
FFT	8c8f1p 1.37	8c8f0p 61	8c4f2p 1.3	8c8f1p 1.98	8c8f0p 96	8c8f1p 1.9	16c16f1p 1.99	16c16f0p 56	16c4f2p 1.1	16c16f1p 2.74	16c16f0p 78	16c4f2p 1.7
FIR	8c8f1p 1.88	8c8f0p 97	8c4f0p 1.9	8c8f1p 3.57	8c8f0p 162	**8c4f1p 3.5**	16c16f1p 3.08	**16c16f0p 99**	16c8f0p 1.8	**16c16f1p 5.92**	**16c16f0p 167**	16c4f1p 3.3
IIR	8c8f1p 0.94	8c8f0p 45	8c4f2p 1.0	8c8f1p 1.55	8c8f0p 72	8c4f2p 1.6	16c16f1p 0.98	16c16f0p 28	16c4f2p 0.7	16c16f1p 1.71	16c16f0p 49	16c4f2p 1.2
K-MEANS	8c8f1p 1.68	8c8f0p 80	8c8f1p 1.6	8c8f1p 2.33	8c8f0p 113	8c8f1p 2.2	16c16f1p 1.50	16c16f0p 43	16c4f2p 1.0	16c16f1p 2.43	16c16f0p 67	16c4f2p 1.5
MATMUL	8c8f1p 1.81	8c8f0p 81	**8c4f1p 1.8**	8c8f1p 3.32	8c8f0p 148	8c4f1p 3.2	16c16f1p 2.86	16c16f0p 80	16c4f2p 1.6	16c16f1p 5.47	16c16f0p 154	16c4f2p 3.1
SVM	8c8f1p 0.77	8c8f0p 37	8c4f2p 0.8	8c8f1p 0.91	8c8f0p 42	8c2f1p 1.0	16c16f1p 1.19	16c16f0p 35	16c4f2p 0.7	16c16f1p 1.47	16c16f0p 42	16c4f2p 1.0
Norm. avg	8c8f1p 0.48	8c8f0p 0.43	8c4f1p 0.30	8c8f1p 1.00	8c8f0p 1.00	8c4f1p 0.94	16c16f1p 0.41	16c16f0p 0.43	16c4f2p 0.29	16c16f1p 1.00	16c16f0p 1.00	16c4f2p 0.97

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. 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) 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 × *float16*, 2 × *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 the loop level with static scheduling of the iterations on the available cores. This policy guarantees 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 synchronization 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 parallelism. To alleviate this limitation, we have adopted a technique based on a block formulation of recursive filters tailored for vector units [49]. 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 Performance, energy efficiency, and area efficiency

We have performed extensive benchmarking considering different configurations of the transprecision cluster and multiple benchmark variants. The number of configurations is limited to a restricted range due to system requirements, depending on the application domain and target technology. Then, once the SoC is fabricated, further tuning can be performed through DVFS, in the specific case in the range between 0.65 V and 0.8 V. The benchmarks are as heterogeneous as possible in terms of application requirements and
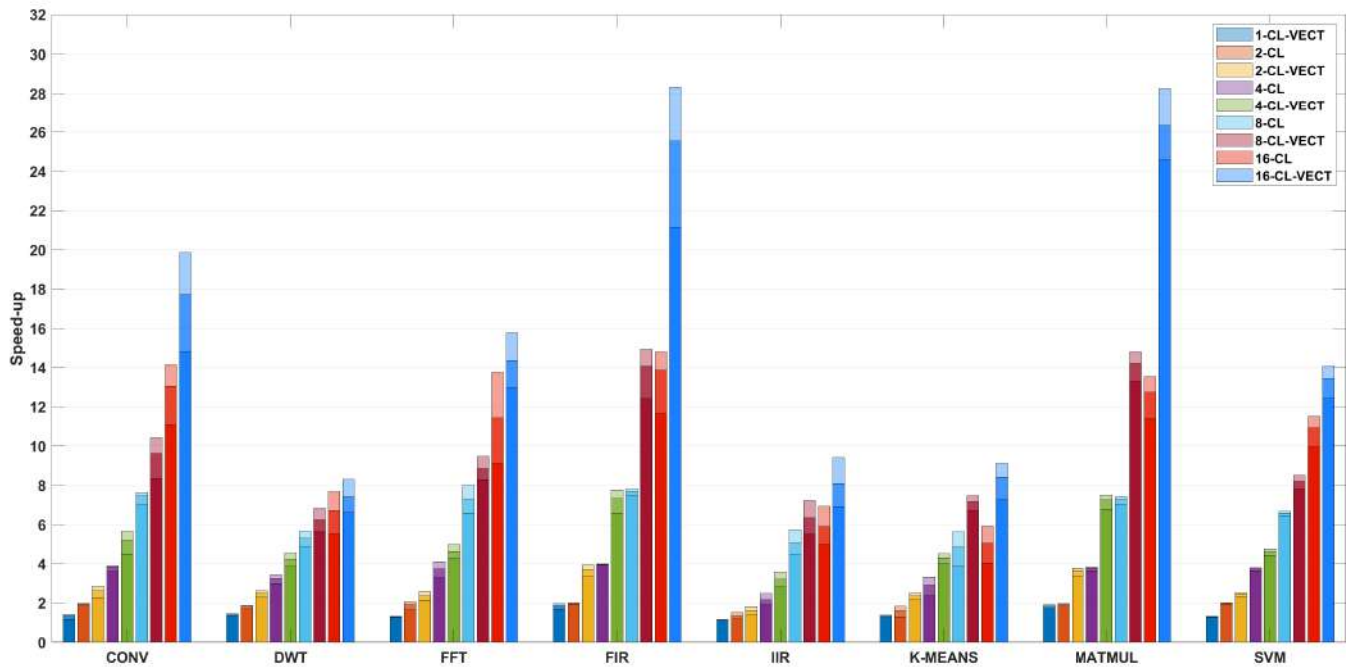
Fig. 6. Speed-ups obtained executing scalar and vector variants on all the platform configurations. Each configuration reports the number of available cores and the support to vectorization. Each bar shows the minimum (dark color), maximum and average (light color) value.

parallel patterns, intending to stress the platform during tests.

We have measured the performance (Gflop/s), the energy efficiency (Gflop/s/W), and the area efficiency (Gflop/s/mm^2) for each benchmark variant and platform configuration: Table 4 reports the results of these experiments. The last row reports the average of the measures, with normalized values in the range between 0 and 1. Computing these metrics allows establishing the configurations that guarantee the best performance and energy/area efficiency for the considered benchmarks.

The configuration with 16 cores, private FPUs, and one pipeline stage (16c16f1p) 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 (16c16f0p), 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 prevents the occurrence of FPU stalls, which are detrimental to energy efficiency.

The configuration with 8 cores and 4 shared FPUs configured with one pipeline stage (8c4f1p) is on average the most area-efficient, with a maximum of 1.8 Gflop/s/mm^2 for scalars and 3.5 Gflop/s/mm^2 for vectors. This configuration

saves area by reducing the number of cores and the sharing factor, but maintaining a single pipeline stage represents the best tradeoff 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 the 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× and 2× – 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 principally evident for FIR, IIR, MATMUL, and KMEANS when passing from 8CL-VECT (8 cores working on vectors) to 16CL (8 cores
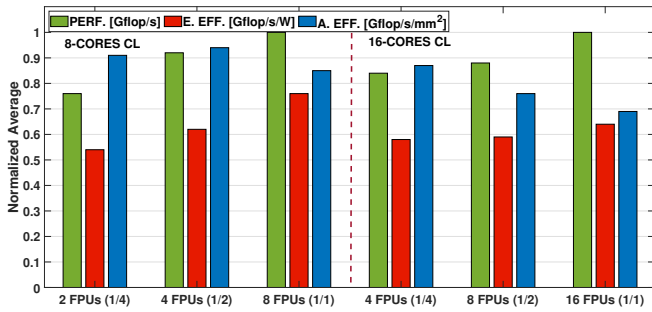
Fig. 7. Performance (PERF.), energy efficiency (E. EFF.), and area efficiency (A. EFF.) fixing one pipeline stage and varying the number of FPUs. The values are the average of the normalized results.
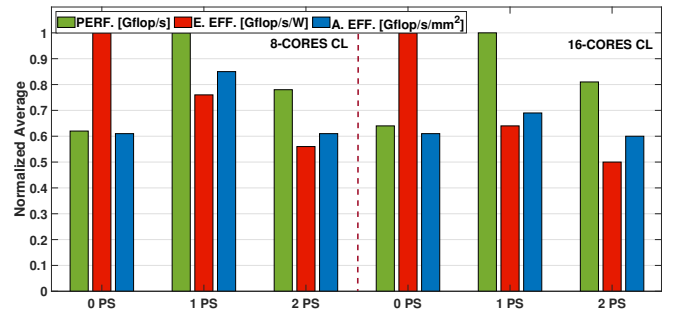

Fig. 8. Performance (PERF.), Energy efficiency (E. EFF.), and area efficiency (A. EFF.) fixing a 1/1 sharing factor and varying the pipeline stages (PS). The values are the average of the normalized resultss.

working on scalars). This effect is due to the different overheads related to parallelization and vectorization. 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 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 has a minor relevance 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 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 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 a further area increase for a second stage is not equally convenient. This trend has a minor impact on 16-cores configurations since the contribution of pipeline logic becomes negligible.

5.4 Case study: CCA-based Brain-Computer Interface

To evaluate the impact of changing the numerical representation on an end-to-end application, we analyze the vectorization of a processing chain based on Canonical Correlation Analysis (CCA) executed on a benchmark dataset [51]. In particular, we focus our analysis on area efficiency to target viable implementations on devices with severe constraints in terms of energy and size (e.g., wearable biomedical systems). We use CCA to emulate the extraction of steady-state visually evoked potential (SSVEP) from Electroencephalogram (EEG), as it is a well-established method to detect the brain response to a given visual stimulus.

This application represents an example of a computationally-intensive processing chain since neural

TABLE 5
Comparison with state-of-the-art architectures in high-performance and low-power embedded domain.

	Hwacha [50]	**Snitch** [44]	**NTX** [43]	**Ariane** [43]	**Xavier**[*]	**Transpire** [39]	**STM32H7**[†]	**Mr.Wolf** [1]	**This work** Best perf. (16c16f1p) Best en. eff. (16c16f0p) Best area eff. (8c4f1p)
Domain	HPC	HPC	HPC	High-End	High-End	IoT	IoT	IoT	IoT
Technology	16nm FinFET	GF 22FDX	GF 22FDX	GF 22FDX	TSMC 12FFN	GF 22FDX	40nm CMOS	40nm CMOS	GF 22FDX
Voltage (V)	$0.55/1^1$	0.80^2	0.80^1	0.80^1	0.75^1	0.80^2	$1.80 / 1.80^1$	$0.8 / 1.10^1$	$0.80 / 0.65 / 0.80^3$
Frequency (GHz)	1.44	1.06	1.55	0.92	1.38	0.2	0.20 / 0.48	0.45	0.37 / 0.30 / 0.43
Area (mm^2)	24	0.89	0.56	0.39	11	1.4	–	10	2.1 / 1.8 / 0.97
Performance^5 (Gflop/s)	280^4	14.4	18.3	2	83	1.08^4	0.05 / 0.13	1.8	**2.9** / 2.3 / 1.7
Energy eff^5 (Gflop/s/W)	108^4	128	135	40	38	96^4	1.2 / 0.9	50	67 / **81** / 68
Area eff^5 (Gflop/s/mm^2)	4.2^4	25.83	32.6	5.2	4.1	1.1^4	- / -	1	1.5 / 1.4 / **1.8**
FP formats	double float float16	double float	float[‡]	float float16 bfloat16 minifloat	float float16	float16 mini-float	float	float	float float16 bfloat16
Programming interface	ISA extension	ISA extension	Memory-mapped configuration	ISA extension	Base ISA	Base ISA	Base ISA	Base ISA	ISA extension
Execution model	SIMT vector-thread unit (accelerator)	Loop-buffers for tensor streaming (accelerator)	Loop-buffers for tensor streaming (accelerator)	SIMD processor	SIMT vector-thread unit (accelerator)	Co-Processor	Processor	Multi-core processor	Multi-core processor
Compiler support	Yes (OpenCL)	Partial (inline ASM)	No	Yes	Yes (CUDA)	Yes	Yes	Yes	Yes

[*] Numbers extracted from [44].   [†] 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.   [4] Original performance is half precision, normalized to binary32.   [5] Performance and efficiency metrics are scaled to 22nm technology node. $1/\lambda$ is considered for performance, $1/\lambda V^2$ scaling is considered for energy efficiency, $1/\lambda^2$ is considered for area efficiency. 0.8V corner is considered for performance and energy efficiency, 0.65V corner is considered for energy efficiency.   [‡] Higher internal accumulation precision with float results.

data processing requires particular effort due to their low Signal-to-Noise ratio [52]. Moreover, computational paradigms for these applications are moving faster towards near-sensor and edge computing, even though they need high accuracy and low numerical errors. As a result, they represent a crucial challenge and a valid use case to explore novel optimizations in adaptive numerical representations at high energy efficiency.

The CCA algorithm (Fig. 9) computes the canonical correlation values between a signal input matrix (i.e., EEG) and an array of reference signals. We have based the CCA implementation on the Golub-Reinsch [53] algorithm due to its computational efficiency and high scalability. As shown in Fig.9, we apply a 5-taps low pass IIR filter to reduce the high-frequency noise, performing then data downsampling to reduce the computational time. The last stage of the preprocessing includes a 3-taps high pass filter to remove DC offset. After these steps, we compute the CCA on the output matrix, which is composed of a QR-factorization (based on Householder rotations), Singular Value Decomposition (SVD), and the extraction of the canonical coefficients (CCs). Finally, by calculating the Euclidean norm on CCs, we determine the level of correlation between input and reference to detect the SSVEP. If the output exceeds a given threshold, a class is assigned. Otherwise, the rest state is asserted.

To assign a precision to program variables (precision tuning), we used the iterative method described in [54], specifying a target accuracy in terms of a maximum mean square error (1^{-5}). The output of the tool is a list of values containing the minimum precision and the maximum range for each variable in the program. From this list, we

TABLE 6
CCA performance.

	MSE†	**Vectorial speed-up**	**2-cores speed-up**	**4-cores speed-up**	**8-cores speed-up**
CCA	3.4^{-6}	$1.3\star$	1.7	3.2	4.3

$\star$ PP 1.7×, CCA+FE 1.2×   † Mean Square Error

associated a corresponding type (*float*, *bfloat16*, or *float16*).

Table 6 reports the execution time (active cycles) of the program parts. We considered two main algorithmic steps: the Preprocessing (PP), which includes LP filter, downsampling, and HP filter; the Canonical Correlation Analysis + Feature Extraction (CCA+FE), including QR-factorization, SVD, CC Extraction, and Euclidian Norm. In the PP step, we obtain 70% lower cycle count with vector execution w.r.t. Scalar version and 20% lower cycle count in the CCA+FE step. Hence, considering the entire processing chain, we have gain 30% performance. Finally, we assessed that the MSE is less than the target value set for precision tuning.

6 COMPARISON WITH THE SoA

Table 5 depicts a comparison with SoA platforms with FP support in high-performance and embedded domains, where performance and efficiency numbers are normalized to 22nm technology node to ease the readability of the results. The number of FP operations has been measured by executing a single-precision matrix multiplication on all the platforms. We chose this kernel since it is a standard benchmark and its performance and power measurements are available for all the considered architectures. Moreover, this benchmark is embarrassingly parallel and can be highly
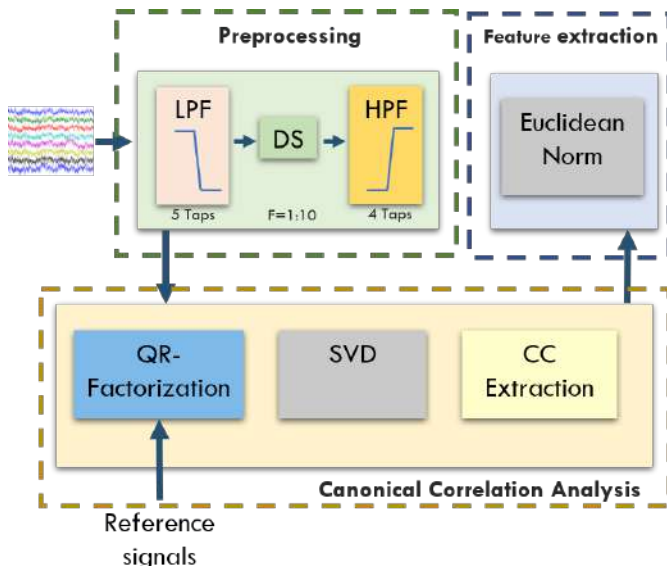
Fig. 9. CCA processing chain for SSVEP systems.

smaller timing pressure on the floating-point units leading to a higher frequency, smaller area and smaller power. It should be noted that the maximum frequency of our work is computed in the worst-case corner, while the Mr.Wolf frequency is measured on silicon, which is typically 20% to 30% faster than the worst-case corner. The second reason for the better performance metrics is related to one of the key contributions of this work: the exploration of the parameters of the cluster to target a specific performance metric (throughput, energy efficiency, or area efficiency). As shown in Table 4 adapting the architectural parameters of the cluster can have a huge impact on the different metrics (up to 190% for performance, up to 42% for energy efficiency, up to 40% for area efficiency), leading to up to $1.4\times$ better performance, $1.6\times$ better energy efficiency, and $1.8\times$ better area efficiency compared to Mr.Wolf.

Finally, compared to most energy-efficient solutions in Table 5, 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-sensor computing, providing a full specification of its main components and a full software stack to execute end-to-end applications. We have performed a design space exploration on an FPGA emulator varying the number of cores, the number of FPUs, and the pipeline stages. A set of experiments on near-sensor algorithms and an analysis based 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 FPUs are the best solution in terms of performance and energy efficiency, outperforming the current state-of-the-art, while a cluster with 8 cores and 4 shared FPUs 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. Overall, 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 valuable insight to system designers and engineers, and the guidelines derived by this exploration can steer the design of future near-sensor computing platforms. Finally, the discussed use case demonstrates that this platform can be effectively used in real-life applications characterized by strict accuracy constraints by leveraging transprecision computing techniques.

optimized on all these architectures. In the last column, we report values for three configurations of the transprecision cluster introduced in this work, corresponding to the best performance, the best energy efficiency, and the best area efficiency. Comparing with high performance embedded platform, 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 $13\times$ 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 $2\times$ the number of operations performed per cycle and is highly beneficial.

With respect to Mr.Wolf, the architecture more closely related to the proposed cluster, our work delivers significantly better performance when comparing to 32-bit floating-point matrix multiplication. One of the reasons for the better metrics is related to the more optimized interconnect and sharing mechanism, granting high functional performance (i.e., low contention on the shared units), as well as a

References

[1] 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.

[2] B. Barrois and O. Sentieys, "Customizing fixed-point and floating-point arithmetic—a case study in k-means clustering," in *2017 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2017, pp. 1–6.

[3] 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.

[4] 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.

[5] D. Zoni, A. Galimberti, and W. Fornaciari, "An FPU design template to optimize the accuracy-efficiency-area trade-off," *Sustainable Computing: Informatics and Systems*, vol. 29, pp. 1–10, 2021.

[6] D. Zuras, M. Cowlishaw, 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] 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.

[8] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International conference on machine learning*, 2016, pp. 2849–2858.

[9] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough, "Precimonious: Tuning assistant for floating-point precision," in *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2013, pp. 1–12.

[10] D. Efnusheva, A. Cholakoska, and A. Tentov, "A survey of different approaches for overcoming the processor-memory bottleneck," *International Journal of Computer Science and Information Technology*, vol. 9, no. 2, pp. 151–163, 2017.

[11] J. N. Coleman, E. Chester, C. I. Softley, and J. Kadlec, "Arithmetic on the European logarithmic microprocessor," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 702–715, 2000.

[12] J. L. Gustafson, *The End of Error: Unum Computing*. CRC Press, 2017.

[13] 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.

[14] 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.

[15] S. Tiwari, N. Gala, C. Rebeiro, and V. Kamakoti, "PERI: A Configurable Posit Enabled RISC-V Core," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 3, pp. 1–26, 2021.

[16] A. Bocco, Y. Durand, and F. de Dinechin, "Hardware support for UNUM floating point arithmetic," in *2017 13th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*. IEEE, 2017, pp. 93–96.

[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 Multiformat Floating-Point Unit Architecture for Energy-Proportional Transprecision Computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 774–787, 2021.

[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] G. Mitra, B. Johnston, A. P. Rendell, E. McCreath, and J. Zhou, "Use of SIMD vector operations to accelerate application code performance on low-powered ARM and Intel platforms," in *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*. IEEE, 2013, pp. 1107–1116.

[25] G. Mitra, E. Stotzer, A. Jayaraj, and A. P. Rendell, "Implementation and optimization of the OpenMP accelerator model for the TI Keystone II architecture," in *International Workshop on OpenMP*. Springer, 2014, pp. 202–214.

[26] RISC-V Foundation, "P Extension." [Online]. Available: https://github.com/riscv/riscv-p-spec

[27] R. M. Russell, "The CRAY-1 computer system," *Communications of the ACM*, vol. 21, no. 1, pp. 63–72, 1978.

[28] 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.

[29] T. Yoshida, "Fujitsu high performance CPU for the Post-K Computer," in *Hot Chips*, vol. 30, 2018.

[30] RISC-V "V" Specification. [Online]. Available: https://github.com/riscv/riscv-v-spec

[31] 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 *ESSCIRC 2014-40th European Solid State Circuits Conference (ESSCIRC)*. IEEE, 2014, pp. 199–202.

[32] D. H. Bailey, H. Yozo, X. S. Li, and B. Thompson, "ARPREC: An arbitrary precision computation package," *Lawrence Berkeley National Laboratory*, 2002.

[33] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, 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.

[34] F. Johansson, "Arb: efficient arbitrary-precision midpoint-radius interval arithmetic," *IEEE Transactions on Computers*, vol. 66, no. 8, pp. 1281–1292, 2017.

[35] V. Lefèvre, "Correctly rounded arbitrary-precision floating-point summation," *IEEE Transactions on Computers*, vol. 66, no. 12, pp. 2111–2124, 2017.

[36] 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.

[37] Lattice sensAI Stack. [Online]. Available: https://www.latticesemi.com/sensAI

[38] 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.

[39] R. Prasad, S. Das, K. J. M. Martin, and P. Coussy, "Floating Point CGRA based Ultra-Low Power DSP Accelerator," *Journal of Signal Processing Systems*, pp. 1–13, 2021.

[40] Arm Helium Technology. [Online]. Available: https://www.arm.com/why-arm/technologies/helium

[41] Arm Cortex M55 processor. [Online]. Available: https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m55

[42] 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

[43] 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.

[44] F. Zaruba, F. Schuiki, T. Hoefler, and L. Benini, "Snitch: A tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execu-

tion of Floating-Point Intensive Workloads," *IEEE Transactions on Computers*, 2020.

[45] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: A design alternative for cache on-chip memory in embedded systems," in *Proceedings of the Tenth International Symposium on Hardware/Software Codesign. CODES 2002 (IEEE Cat. No. 02TH8627)*. IEEE, 2002, pp. 73–78.

[46] PULP GitHub page. [Online]. Available: https://github.com/pulp-platform

[47] Auto-vectorization in GCC. [Online]. Available: https://www.gnu.org/software/gcc/projects/tree-ssa/vectorization.html

[48] GNU Foundation, "libgomp runtime." [Online]. Available: https://gcc.gnu.org/onlinedocs/libgomp/

[49] 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.

[50] C. Schmidt, J. Wright, Z. Wang, E. Chang, A. Ou, W. Bae, S. Huang, A. Flynn, B. Richards, K. Asanović, E. Alon, and B. Nikolić, "4.3 An Eight-Core 1.44GHz RISC-V Vector Machine in 16nm FinFET," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 58–60.

[51] M. Salvaro, S. Benatti, V. J. Kartsch, M. Guermandi, and L. Benini, "A minimally invasive low-power platform for real-time brain computer interaction based on canonical correlation analysis," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 967–977, 2019.

[52] F. Lin, J. K. Zao, K. Tu, Y. Wang, Y. Huang, C. Chuang, H. Kuo, Y. Chien, C. Chou, and T. Jung, "SNR analysis of high-frequency steady-state visual evoked potentials from the foveal and ex-trafoveal regions of Human Retina," in *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2012, pp. 1810–1814.

[53] A. A. Maciejewski and C. A. Klein, "The singular value decomposition: Computation and applications to robotics," *The International journal of robotics research*, vol. 8, no. 6, pp. 63–79, 1989.

[54] 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.

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 bio-signal 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.

Simone Benatti received the Ph.D. degree in Electrical Engineering and Computer Science from the University of Bologna, in 2016. He currently holds a postdoctoral position with the University of Bologna. His research interests include energy efficient embedded wearable systems for advanced Human Computer Interaction and algorithms for edge computing. In this field, he has authored/coauthored more than 40 papers in international peer-reviewed conferences and journals. Previously, he worked 8 years as an Electronic Designer and R&D Engineer of electromedical devices.

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-programmable embedded architectures.

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 University of 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 Academia Europaea. He is the recipient of the 2016 IEEE CAS Mac Van Valkenburg Award and the 2020 EDAA Achievement 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 O. Pederson Best Paper Award 2018, 2020 IEEE TCAS Darlington Best Paper Award, 2020 IEEE TVLSI Prize Paper Award.

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 emerging computing architectures.