TRANSPIRE: An energy-efficient TRANSprecision floating-point Programmable archItectuRE

Rohit Prasad, Satyajit Das, Kevin Martin, Giuseppe Tagliavini, Philippe Coussy, Luca Benini, Davide Rossi

TRANSPIRE: An energy-efficient TRANSprecision floating-point Programmable archItectuRE

Rohit Prasad*†, Satyajit Das§, Kevin J. M. Martin*, Giuseppe Tagliavini†, Philippe Coussy*, Luca Benini†‡ and Davide Rossi†

*Univ. Bretagne-Sud, UMR 6285, Lab-STICC, F-56100 Lorient, France, [firstname].[lastname]@univ-ubs.fr,

†Electrical, Electronic, and Information Engineering, University of Bologna, Italy, [firstname].[lastname]@unibo.it

‡Integrated Systems Laboratory, ETH Zurich, Switzerland, [first-initial][last name]@iis.ee.ethz.ch

§Department of Computer Science and Engineering, IIT Palakkad, India, satyajitdas@iitkpd.ac.in

***Abstract*—In recent years, Coarse Grain Reconfigurable Architecture (CGRA) accelerators have been increasingly deployed in Internet-of-Things (IoT) end nodes. A modern CGRA has to support and efficiently accelerate both integer and floating-point (FP) operations. In this paper, we propose an ultra-low-power tunable-precision CGRA architectural template, called TRANSprecision floating-point Programmable archItectuRE (TRANSPIRE), and its associated compilation flow supporting both integer and FP operations. TRANSPIRE employs transprecision computing and multiple Single Instruction Multiple Data (SIMD) to accelerate FP operations while boosting energy efficiency as well. Experimental results show that TRANSPIRE achieves a maximum of $10.06\times$ performance gain and consumes $12.91\times$ less energy w.r.t. a RISC-V based CPU with an enhanced ISA supporting SIMD-style vectorization and FP data-types, while executing applications for near-sensor computing and embedded machine learning, with an area overhead of $1.25\times$ only.**

I. Introduction

In this emerging era of the Internet of Things (IoT), there is an ever-increasing demand for ultra-low-power and energy-efficient computing architectures. In this scenario, most of the promising approaches to improve performance and energy efficiency of these platforms exploit parallelism [23], reconfigurability [6], and heterogeneity [11] to fit with the workloads for near-sensor IoT end nodes. Studies of more than a decade have shown that Coarse Grain Reconfigurable Architectures (CGRAs) can provide silicon efficiency approaching that of Application-Specific Integrated Circuit (ASIC) by exploiting spatial computation typical of dedicated hardware while keeping programmability typical of instruction processors [5]. CGRAs were studied at first for accelerating the inner loops computation of the kernels. Over the years, they have evolved and are now competitive solutions in the domain of high-performance accelerators [10], [16], [22]. Recently, the research community is pushing CGRAs towards ultra-low-power low-end applications, such as near-sensor processing and low power wearable applications [8], [11]. While recent CGRA architectures demonstrated leading energy efficiency when executing fixed-point workloads, floating-point (FP) support is becoming a must for emerging IoT end nodes [1]. Although fixed-point representation has some clear advantages in terms of energy per operation due to the simpler architecture of integer arithmetic units over FP counterparts [18], the latter is much more attractive, especially in near-sensor processing domains such as bio-potentials, often leveraging linear algebra algorithms featuring extremely high dynamic range [20], but also in other fields, such as audio and robotics [13]. First, porting fixed-point applications to FP is not always neutral from a numerical stability viewpoint, and is often a time-consuming task requiring an in-depth understanding of the applications. Moreover, even when it is straightforward to transform FP code into its fixed-point counterpart, this is not necessarily the best solution energy-wise, since the adjustments and normalization instructions required to deal with the dynamic range might incur significant overhead [20].

An emerging approach to reduce the power consumption of FP operations while preserving the dynamic required by applications without manual adjustments is *transprecision computing* [24]. This paradigm aims at designing systems which deliver the required precision for intermediate computations given an accuracy bound specified by the user, and leverages automated tools to associate reduced-precision types to program variables [24]. An attempt towards transprecision computing was made by introducing two new custom FP data-types (*binary16alt* and *binary8*) and a hardware unit called smallFloat Unit (SFU) [18], which employs IEEE-754 *binary32*, *binary16*, and two new data-types, namely *binary16alt* (featuring a higher dynamic range vs. *binary16*) and *binary8*. Exploiting these data types leads to significant improvement in terms of performance and energy efficiency [18].

This work combines the principles of transprecision computing with the flexibility of CGRA in exploiting multi-datapath for high Instruction Level Parallelism (ILP) and Data Parallelism (DP), to propose a high energy efficiency low power FP-CGRA architecture. The proposed CGRA gains $10.06\times$ performance and consumes $12.91\times$ less energy over a RISC-V CPU extended with SIMD-style vectorization and executing same kernels using same FP data-types as of the proposed CGRA. In this context, the contributions are:

- A heterogeneous CGRA architecture supporting both integer and FP data types, and employing transprecision computing, multi-cycle operations, and SIMD;
- Its associated compilation flow to efficiently exploit parallelism between the FP operations at instruction level leveraging static scheduling;
- A set of experimental results which provide comparisons

with different state-of-the-art architectures for performance, energy consumption, and efficiency.

The rest of the paper is organized as follows. Section II discusses the state-of-the-art architectures for FP acceleration in CGRAs. Section III discusses the proposed architecture and its compilation flow. Section IV shows experimental results and comparisons. Finally, section V provides a conclusion.

II. Related Work

Typically CGRAs are systolic arrays containing a large number of PEs with simple interconnects. Thanks to this design, they can efficiently exploit both instruction-level parallelism (ILP) and data parallelism (DP). CGRAs have been designed as accelerators coupled to a host CPU [6]; however, the proposed solutions lack support for FP operations. Research works have discussed the role of CGRAs to employ approximate computing and SIMD, or even the adoption of multiple approximation modes to further exploit DP. These proposals also lack support for FP operations [3], [11].

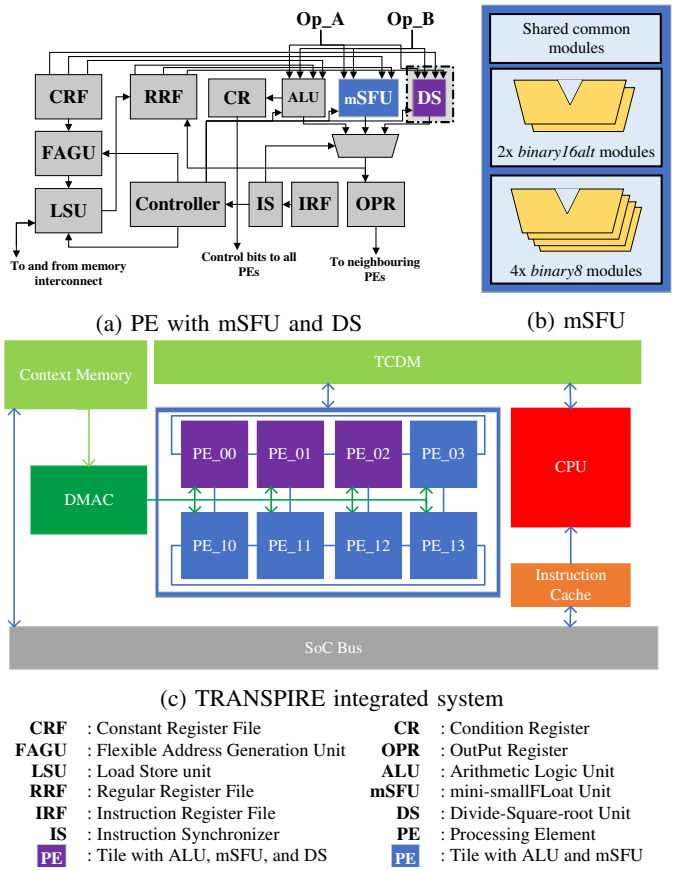
In the past, very few works have been presented where CGRAs are supporting FP operations because adding support for FP operations imposes many restrictions on the architecture. In FloRA [15] and Wave CGRA [21], integer-based PEs are combined for computing FP operations. This design results in low power consumption, but it increases the width of interconnect and degrades the output quality. In [4], Butter array reuses the integer-based adder and multiplier with additional packing and rounding units for computing FP multiplication and addition. These additional units result in significant area overhead. Overall, these architectures lack native support for FP operations. In [12], a Stream Dual-Track-CGRA (SDT-CGRA) with flexible interconnects is presented, to support different computation models and to reduce the hardware complexity. SDT-CGRA converts all FP input data into fixed-point before computing the output; this demands extra overhead which comes from these conversions. A common peculiarity in all of these architectures is that the power consumption required to support FP operations is too demanding for their adoption in IoT end nodes or as ultra-low-power architectures. Recent trend is to equip IoT platforms e.g., microcontroller units like M4 and M7 [2] with FP unit, this is because with the scaling of technology below 40nm, the cost of an FP operation is getting near 1pJ/op [19], [25], so it has become affordable in terms of absolute power to use FP in IoT.

In view of these works, we exploit for the first time, to the best of authors' knowledge, a static mapping approach to natively support FP operations in CGRA together with transprecision computing to maintain the energy consumption in the ultra-low-power domain.

III. TRANSPIRE and Compilation Flow

A. Architecture

1) CGRA Integrated System: Fig. 1c shows the configuration of TRANSPIRE. The integrated system consists of a 4×2 heterogeneous processing element (PE) array, a DMA controller, and a context memory. TRANSPIRE is loosely



(a) PE with mSFU and DS (b) mSFU

(c) TRANSPIRE integrated system

Fig. 1: TRANSPIRE integrated system and PE with mSFU and DS

coupled with a host CPU, enabling it to execute a complete kernel independently. Both TRANSPIRE and the CPU share data through a Tightly Coupled Data Memory (TCDM), which consists of multiple memory banks connected through a low-latency logarithmic interconnect. Before execution, instructions and constants (i.e., configuration data) of each PE are stored in the context memory, and then the DMA controller sends these data to their respective PEs.

2) Heterogeneous PEs: The PEs are connected through a mesh torus network for sharing data with adjacent PEs and a bus network for context broadcast. Each PE includes an ALU for integer-based operations and a mini-smallFloat unit (mSFU) for FP operations.

Taking into consideration that divide and square-root operations have limited occurrence, a Divide-Square-root (DS) unit is introduced in the first three tiles to restrict the unnecessary data movements. For example, in equation $out = (a \div b) * (x + \sqrt{y})$, the computations of $(a \div b)$ and $(x + \sqrt{y})$ can execute in parallel. If all PEs had a DS unit, these parallel operations could be mapped into non-adjacent PEs (say PE_00 and PE_12) and final multiplication operator onto PE_00. To execute the final multiplication, data from PE_12 would be moved into PE_00 requiring 3 $MOVE$ operations. Conversely, in the proposed PE arrangement choice and assuming

Operator	Latency (cycles)	Shared/ Private	Data-type	mSFU/ DS
float-absolute	1	Shared	IEEE-754 *binary32*	mSFU
float-less-than	1	Shared	IEEE-754 *binary32*	mSFU
float-add	2	Private	*binary16alt, binary8*	mSFU
float-sub	2	Private	*binary16alt, binary8*	mSFU
float-mul	2	Private	*binary16alt, binary8*	mSFU
float-divide	5	Private	*binary16alt*	DS
float-square-root	5	Private	*binary16alt*	DS

TABLE I: FP operators in mSFU and DS.

	DS_mSFU (μm^2)	**DS_SFU** (μm^2)
Total	1,031	5,395

TABLE II: Total cell area of DS in TRANSPIRE and RI5CY [18].

the worst case, those parallel operations are mapped onto PE_00 and PE_02, and the final multiplication onto PE_00. To perform the final multiplication data has to be moved from PE_02 to PE_00, which requires 2 $MOVE$ operations only.

Each PE has a Constant Register File (CRF) to store immediate values. A Regular Register File (RRF) and an OutPut Register (OPR) store the temporary values. The Instruction Synchronizer (IS) synchronizes the instructions; it is flexible and currently supports 1-cycle, 2-cycles, and 5-cycles operations. After the IS has issued a fetch enable signal, the Controller fetches the instructions from the Instruction Register File (IRF). Load and Store instructions carry the CRF addresses of the data required by the Flexible Address Generation Unit (FAGU) for calculating the addresses. These addresses are provided to the Load-Store Unit (LSU), which performs memory accesses. The Jump Register (JR) stores the target address of the jump instructions. In the case of $cjump$ (conditional jump) instructions, which include two addresses, the true path is evaluated in the JR applying a boolean OR on the bits of the Condition Register (CR).

3) mSFU: An mSFU includes 2 slices of *binary16alt* units and 4 slices of *binary8* units. The datapath is 32-bits wide, which enables TRANSPIRE to perform SIMD operations for custom FP data types. The operators in the mSFU are non-blocking and non-pipelined. Float-absolute and float-less-than operators support the IEEE-754 *binary32* format and are shared among these slices in mSFU. Table I lists the employed FP operators and their latency.

4) DS unit: The DS unit brings support for *binary16alt*-based divide and square-root operators. DS units have been stripped down from divide-square-root unit of SFU [18] to support only *binary16alt* data-type and only one rounding mode (truncation). Table I lists the latency of the operators. After synthesizing both units in the 28nm FD-SOI process node, it was observed that the DS unit is $5.23\times$ smaller than the divide and square-root unit in SFU which supports 4 FP data types (i.e., IEEE-754 *binary64*, *binary32*, *binary16*, *binary16alt*, and *binary8*) and 4 rounding modes. Inherently, divide and square-root operators reuse the common sub-modules, that enabled TRANSPIRE to include three DS units without bringing significant area overhead. Total cell area comparison is shown in Table II.

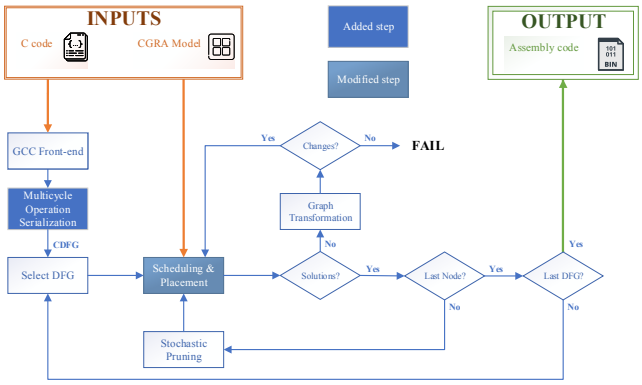


Fig. 2: Simplified view of the proposed Compilation Flow

B. Compilation Flow

The compilation flow exploits the GCC front-end to get the intermediate representation of the application code. The original input C code needs to be modified to:

1) support hardware-based address generation in FAGU; every variable is updated with global index representation i.e.,
   $Variable[(i+A)*(j+B)][(k+C)*(l+D)]$
   where i, j, k, l are loop variables and A, B, C, D are constants or variables;
2) exploit multi-datapath in CGRA and ease the mapping process of the application for high ILP;
3) differentiate *binary16alt* and *binary8* operations.

It is noteworthy that it only took a few hours to modify all the kernels [IV-A] taken into account in this paper.

TRANSPIRE is modeled as a bipartite directed graph with operator and register nodes. An application is modeled as a Control Data Flow Graph (CDFG), which is itself comprised of a Control Flow Graph (CFG) within which each Basic Block (BB) represents a Data Flow Graph (DFG). A DFG is a bipartite directed acyclic graph composed of operation and data nodes. Arrows connecting the nodes in DFGs represent data dependencies. The homomorphism between TRANSPIRE and DFG makes the mapping of CDFG onto TRANSPIRE a sub-graph finding problem. Fig. 2 represents the steps involved in the proposed compilation flow.

After CDFG formation, each operation node in DFGs is also checked for multi-cycle operations. If a node represents a multi-cycle operation, then a graph transformation is performed by adding as many dummy nodes as the number of cycles required to perform that particular operation; for instance, if an operation requires 5 cycles, then 4 dummy nodes are added below that specific operation node. The transformed graph is then passed to the Scheduling and Binding step. While mapping multi-cycle operations on TRANSPIRE, there are two main challenges to address, as exemplified in the C code shown in figure 3.

1) All consecutive multi-cycle operations (e.g., $fmul$ and $fadd$ in Fig. 4) should be mapped onto the same PE, to eliminate the chances of undesirable $MOVE$ operations;

```
int i, j, k; float a[15], b[15], c[256][23], out;
for (i = 14; i >= 0; i--){
 for (j = 0; j < 23; j++){
  for (k = 0; k < 256; k++){
   out = fadd16alt(a[i], fmul16alt(b[i], c[k][j]));
  }
 }
}
```

Fig. 3: Code sample to highlight the two main challenges. The mapping of the kernel of this loop is shown in Fig. 4

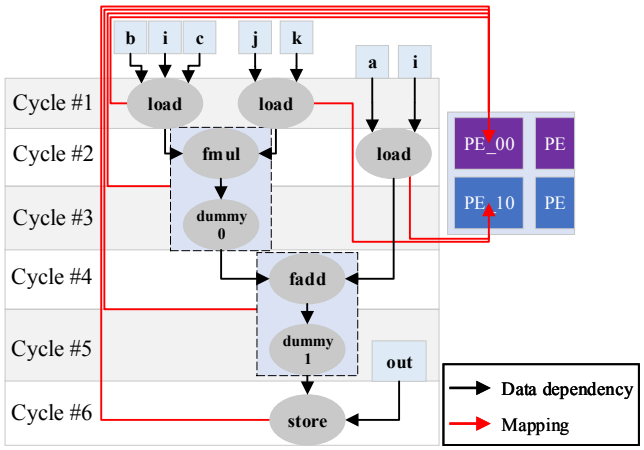


Fig. 4: Mapping of DFG nodes onto PEs.

2) impose minimum restriction on the algorithm in terms of resource availability (i.e., minimizing data routing by mapping nodes which share data onto adjacent PEs).

These challenges are addressed by: (1) carefully removing unwanted nodes between two consecutive multi-cycle operations which might cause $MOVE$ operations; (2) updating the algorithm for resource availability after a PE has been locked for performing multi-cycle operation; (3) immediately unlock that PE for mapping of the next operation, without consuming any extra cycles.

Both scheduling and binding processes are done concurrently; to avoid scaling up of available solutions, stochastic pruning of the solution is performed before proceeding to the next BB in CDFG [7], [9]. If a multi-cycle operation is detected, then only the first encountered node from the multi-cycle nodes chain is mapped using Levi's algorithm [17]; this mapping solution is then copied (static mapping) onto other remaining nodes in the chain. Once all BBs have been mapped, the compiler generates assembly code for a single mapping of the CDFG.

Finally, the assembler combines the assembly code produced by the compiler and the Instruction Set Architecture (ISA) format to generate the machine code used for PEs configuration. This machine code consists of instructions and constants, which are sent to the IRF and CRF of the PEs. Instructions also include the addresses of the indexes sent to the CRF, used by FAGU for address calculation.

Kernel	Operations executed	Highest loop iteration	Input Data size (bits)
mean_covariance	397,348	47,104	94,208
Householder	35,632	1,360	9,216
Accumulate	106,298	1,240	8,704
Diagonalize	74,987	2,368	9,216
PC	168,738	11,776	102,400
CONV	766,728	4,096	131,072
DWT	39,456	448	16,384
SVM	15,630	896	72,000

TABLE III: Complexity of kernels.

IV. Experiments and Results

A. Evaluation Methodology

A set of applications has been chosen for performing the experiments. These applications implement the fundamental algorithms used in two domains relevant for ultra-low-power systems, near-sensor computing and embedded machine learning.

- PCA performs Principal Component Analysis. This algorithm is used for seizure detection, which covers up a wide range of Electroencephalography (EEG) signal processing applications. PCA consists of 5 sequentially executed kernels i.e., Mean Covariance, Householder, Accumulate, Diagonalize, and Principal Component (PC).
- CONV implements a 5×5 convolution kernel. This algorithm is used for image and audio processing applications.
- DWT computes the Discrete Wavelet Transform. This algorithm is used for Electrocardiography (ECG) analysis applications.
- SVM is the prediction stage of a Support Vector Machine. This algorithm is used as a classifier for predicting traffic data, ECG, etc.

The complexity of these kernels is reported in Table III. PCA is further split into five algorithmic kernels.

TRANSPIRE is evaluated against three architectures:

1) RI5CY_FPU [14] is a 4-stage RISC-V CPU with support for IEEE-754 *binary32* FP data-type, for state-of-the-art architecture comparison.
2) RI5CY_SFU is a 4-stage RISC-V CPU with an enhanced ISA supporting SIMD-style vectorization and includes SFU [18], for comparison with state-of-the-art architecture having similar features.
3) TRANSPIRE_FPU is the version of TRANSPIRE which supports IEEE-754 *binary32* FP data-type, for comparison of the same architecture which is IEEE compliant and thus yield no quality degradation in results. Organization of FP operators is the same as it is in TRANSPIRE.

RI5CY [14] is an in-order 4-stage RISC-V CPU which supports SIMD extensions, custom instructions, and misaligned load support; these features extensively reduce the bandwidth requirements for data memory and increase the computational efficiency. The core is highly optimized for DSP benchmarks, so it is a good candidate for a fair comparison with TRANSPIRE.

Kernel	Average deviation (%)	Data-type
mean_covariance	4.80	*binary16alt*
Householder	0.33	*binary16alt*
Accumulate	9.03	*binary16alt*
Diagonalize	5.49	*binary16alt*
PC	1.54	*binary16alt*
CONV	2.32	*binary8*
DWT	6.98	*binary8*
SVM	7.11	*binary8*

TABLE IV: Accuracy performance of TRANSPIRE

	TRANSPIRE	RI5CY SFU	TRANSPIRE FPU	RI5CY FPU
DMA Controller	593	4 KiB Instruction Cache	593	4 KiB Instruction Cache
Interconnect	6,273		6,273	
Context memory	9,345		9,345	
TCDM	65,164		65,164	
PE Array	186,407		174,230	
Total	267,784	213,371	255,605	185,812

TABLE V: Total cell area (μm^2) breakdown and comparison.

B. Quality of Results

Table IV shows the accuracy performance of *binary16alt* and *binary8* compared with IEEE-754 *binary32* and *binary16*, respectively, as they have the same dynamic range. Here, we observe that accuracy loss is 9.03% for Accumulate kernel due to extensive computations on sub-normal FP numbers and 0.33% for Householder kernel due to fewer computations involving sub-normal FP numbers.

C. Implementation Results

For the sake of comparison, we have taken into account three parameters: performance, energy consumption, and area. All experiments have been performed on a post-synthesis netlist. The same parameters have been used for the synthesis of the four architectures, namely a 28nm UTBB FD-SOI process node, 50 MHz frequency, 0.6V operating voltage, worst-case analysis corner (i.e., slow NMOS, slow PMOS, 125°C temperature, and low power low Vt transistors).

The Context Memory of TRANSPIRE is sized at 4 KiB to fit the configuration data (i.e., instructions and constants). The TCDM is sized at 32 KiB with 4 memory banks. TRANSPIRE features 4×2 tiles, and each PE has a 21×64-bits instruction memory, a 20×32-bits constant register file, and a 32×8-bits regular register file. TRANSPIRE_FPU is the version of TRANSPIRE where the mSFU is replaced by a Floating Point Unit (FPU) supporting IEEE-754 *binary32*, and the DS unit is replaced by an IEEE-754 *binary32* compliant divide-square-root unit. Both configurations are big enough to accommodate the binaries of the applications used for benchmarking. The RI5CY_SFU and RI5CY_FPU has 4 KiB of instruction cache and 32 KiB of data memory.

Table V shows a breakdown analysis of the total cell area and performs a comparison of all four architectures. TRANSPIRE is $1.25\times$ bigger than RI5CY_SFU, $1.05\times$ bigger than TRANSPIRE_FPU, and $1.44\times$ bigger than RI5CY_FPU. A complete area breakdown of PE with mSFU and DS is shown in Fig. 5b. It can be observed that mSFU and DS take 9% and 4% of the total cell area of a PE respectively. It can also be observed in Fig. 5a that FPU is only $1.09\times$ bigger than

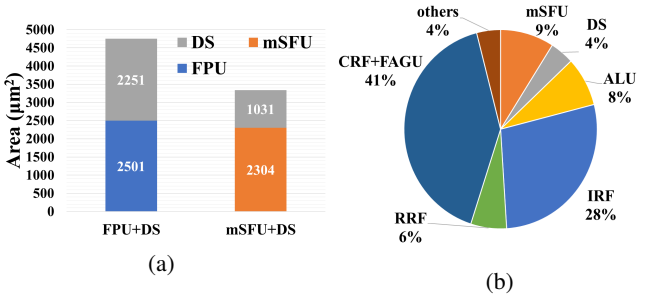


Fig. 5: (a) Total cell area comparison of FPU, mSFU, and DS (b) Area breakdown of PE with mSFU and DS

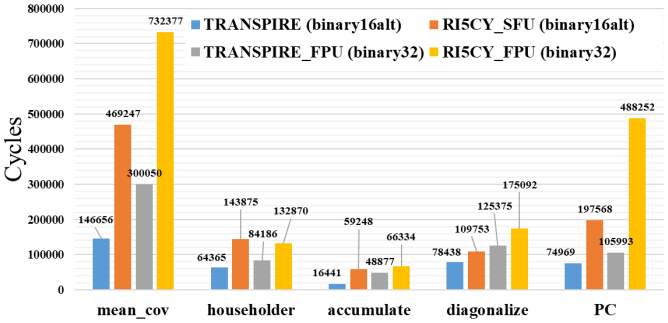


Fig. 6: Performance of PCA kernels (IEEE-754 *binary32* and *binary16alt* data-types).

mSFU and IEEE-754 binary32 compliant divide-square-root unit is $2.18\times$ bigger than DS unit. When combined, FPU+DS is $1.42\times$ bigger than mSFU+DS.

D. Performance Results

Fig. 6 shows the performances of TRANSPIRE (*binary16alt*), RI5CY_SFU (*binary16alt*), TRANSPIRE_FPU (IEEE-754 *binary32*), and RI5CY_FPU (IEEE-754 *binary32*) architectures on running PCA kernels. In Table VI, TRANSPIRE (*binary8*) and RI5CY_SFU (*binary8*) are compared by running CONV, DWT, and SVM kernels. It is evident that TRANSPIRE outperforms the other architectures: (1) TRANSPIRE_FPU does not support multiple SIMD because datapath is 32-bits wide, thus less FP operations are executed per clock cycles; (2) RI5CY_SFU employs a 4-stage pipeline architecture yet fails to surpass the average PE utilization of TRANSPIRE i.e., 72% (PCA), 63% (CONV), 87.5% (DWT), and 47% (SVM). RI5CY_SFU can execute a maximum of 4 parallel FP operations while TRANSPIRE can have a maximum of 32 parallel FP operations with an area overhead of $1.25\times$ only; (3) RI5CY_FPU neither supports SIMD nor surpasses the average PE utilization of TRANSPIRE with its 4-stage

Kernel	TRANSPIRE binary8 (cycles)	RI5CY_SFU binary8 (cycles)	Gain
CONV	268,179	1 455,097	5.43×
DWT	11,140	16,912	1.52×
SVM	11,408	114,747	10.06×

TABLE VI: Performance of TRANSPIRE and RI5CY_SFU.

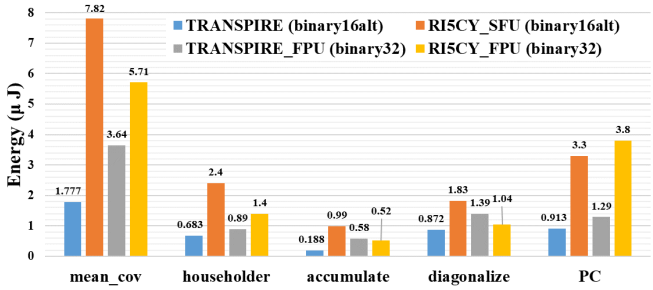


Fig. 7: Energy consumption of PCA kernels (IEEE-754 *binary32* and *binary16alt* data-types).

Kernel	TRANSPIRE binary8 (μJ)	RI5CY_SFU binary8 (μJ)	Gain
CONV	3.036	21.506	7.08×
DWT	0.124	0.256	2.07×
SVM	0.123	1.588	12.91×

TABLE VII: Energy consumption of TRANSPIRE and RI5CY_SFU for CONV, DWT, and SVM.

pipeline architecture. TRANSPIRE achieves a maximum of 10.06× better performance w.r.t. RI5CY_SFU.

E. Energy Consumption

Fig. 7 compares the energy consumption of TRANSPIRE (*binary16alt*) with RI5CY_SFU (*binary16alt*), TRANSPIRE_FPU (IEEE-754 *binary32*), and RI5CY_FPU (IEEE-754 *binary32*) using PCA kernels. Table VII shows a comparison between TRANSPIRE (*binary8*) and RI5CY_SFU (*binary8*) using CONV, DWT, and SVM kernels, here TRANSPIRE consumes 12.91× less energy w.r.t. RI5CY_SFU. It can be observed that TRANSPIRE consumes the minimum energy among these architectures:

- TRANSPIRE_FPU performs non-vectored FP operations on 32-bits wide operands, thus executes more instructions and results in high energy consumption;
- RI5CY_SFU have complex core w.r.t. TRANSPIRE cores, thus greater energy consumption.
- RI5CY_FPU perform non-vectored FP operations on 32-bits wide operands and have complex core w.r.t. TRANSPIRE cores, thus consumes more energy.

Fig. 8 shows a comparison of energy efficiency i.e., Million Operations Per Second Per milliWatt (MOPS/mW) between TRANSPIRE and TRANSPIRE_FPU. Since IEEE-754 *binary32* is comparable with *binary16alt*, we considered PCA kernels only. In the case of Householder and Diagonalize kernels, the energy-efficiency is less compared to the rest because these are high control intensive kernels due to complex control flow constructs. TRANSPIRE reaches a maximum of 224 MOPS/mW and TRANSPIRE_FPU reaches a maximum of 156 MOPS/mw, while RI5CY_SFU and RI5CY_FPU have an overall energy-efficiency of 60 MOPS/mW and 24 MOPS/mW respectively.

V. Conclusion

This paper presents an ultra-low-power CGRA and its associated compilation flow to perform acceleration of FP applications. TRANSPIRE is heterogeneous and employs transprecision computing, multi-cycle operations, and SIMD for natively supporting FP operations. To efficiently accelerate and execute FP operations, TRANSPIRE optimally exploits ILP as well as DP. The compiler uses a static mapping approach to map the FP operations onto PEs. As a result, TRANSPIRE achieves 10.06× better performance gain and 12.91× less energy consumption w.r.t. RISC-V CPU with an area overhead of 1.25× only.

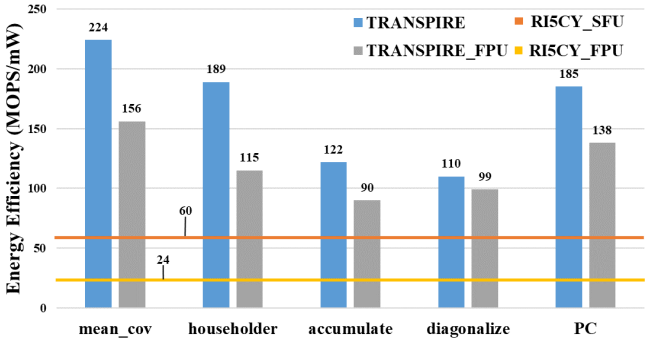


Fig. 8: Energy Efficiency comparison using PCA kernels.

References

[1] STM32L4 MCU series: Excellence in ultra-low-power with performance. *STM32 Ultra Low Power MCUs*, 2018.

[2] STM32H7 Dual Core World Most Powerful MCU. *STM32H7 Series*, June, 2019.

[3] O. Akbari, M. Kamal, A. Afzali-Kusha, M. Pedram, and M. Shafique. PX-CGRA: Polymorphic approximate coarse-grained reconfigurable architecture. In *DATE 2018*, 2018.

[4] C. Brunelli, F. Garzia, D. Rossi, and J. Nurmi. A Coarse-grain Reconfigurable Architecture for Multimedia Applications Supporting Subword and Floating-point Calculations. *J. Syst. Archit.*, 56(1), 2010.

[5] A. Carroll, S. Friedman, B. V. Essen, A. Wood, B. Ylvisaker, C. Ebeling, and S. Hauck. Designing a Coarse-grained Reconfigurable Architecture for Power Efficiency . 2007.

[6] S. Das, K. J. M. Martin, P. Coussy, and D. Rossi. A Heterogeneous Cluster with Reconfigurable Accelerator for Energy Efficient Near-Sensor Data Analytics. In *ISCAS*, pages 1–5, May 2018.

[7] S. Das, K. J. M. Martin, P. Coussy, D. Rossi, and L. Benini. Efficient mapping of cdfg onto coarse-grained reconfigurable array architectures. In *ASP-DAC 2017*, 2017.

[8] S. Das, K. J. M. Martin, D. Rossi, P. Coussy, and L. Benini. An Energy-Efficient Integrated Programmable Array Accelerator and Compilation Flow for Near-Sensor Ultralow Power Processing. *TCAD*, 38(6), 2019.

[9] S. Das, T. Peyret, K. Martin, G. Corre, M. Thevenin, and P. Coussy. A Scalable Design Approach to Efficiently Map Applications on CGRAs. In *ISVLSI*, pages 655–660, July 2016.

[10] B. De Sutter, P. Raghavan, and A. Lambrechts. Coarse-Grained Reconfigurable Array Architectures. In *Handbook of Signal Processing Systems*. Springer US, 2010.

[11] L. Duch, S. Basu, R. Braojos, G. Ansaloni, L. Pozzi, and D. Atienza. Heal-wear: An ultra-low power heterogeneous system for bio-signal analysis. *IEEE CAS I: Regular Papers*, 64(9):2448–2461, Sep. 2017.

[12] X. Fan, D. Wu, W. Cao, W. Luk, and L. Wang. Stream Processing Dual-Track CGRA for Object Inference. *IEEE Transactions on VLSI Systems*, 26(6), 2018.

[13] G. Frantz and R. Simar. Comparing Fixed and Floating Point DSPs. *SPRY061, Texas Instruments*, 2004.

[14] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Grkaynak, and L. Benini. Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices. *IEEE Transactions on VLSI Systems*, 25:27002713, Oct 2017.

[15] M. Jo, D. Lee, K. Han, and K. Choi. Design of a coarse-grained reconfigurable architecture with floating-point support and comparative study. *Integration, the VLSI Journal*, 47, Jan. 2013.

[16] S. Kim, Y. Park, J. Kim, M. Kim, W. Lee, and S. Lee. Flexible video processing platform for 8k UHD TV. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–1, Aug. 2015.

[17] G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9(4):341, Dec. 1973.

[18] S. Mach, D. Rossi, G. Tagliavini, A. Marongiu, and L. Benini. A Transprecision Floating-Point Architecture for Energy-Efficient Embedded Computing. In *ISCAS*, pages 1–5, May 2018.

[19] S. Mach, F. Schuiki, F. Zaruba, and L. Benini. A 0.80 pj/flop, 1.24 Tflop/sW 8-to-64 bit Transprecision Floating-Point Unit for a 64 bit RISC-V Processor in 22 nm FD-SOI. In *VLSI-SOC*, 2019.

[20] F. Montagna, S. Benatti, and D. Rossi. Flexible, Scalable and Energy Efficient Bio-Signals Processing on the PULP Platform: A Case Study on Seizure Detection. *Journal of Low Power Electronics and Applications*, 7(2):16, June 2017.

[21] C. Nicol. A Coarse Grain Reconfigurable Array (CGRA) for statically scheduled data flow computing. *WAVE Computing*, 2016.

[22] C. Nicol. A Dataflow Processing Chip for Training Deep Neural Networks. *WAVE Computing*, 2017.

[23] D. Rossi, I. Loi, F. Conti, G. Tagliavini, A. Pullini, and A. Marongiu. Energy efficient parallel computing on the pulp platform with support for openmp. In *IEEEI 2014*, 2014.

[24] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini. A transprecision floating-point platform for ultra-low power computing. In *DATE 2018*, 2018.

[25] B. Zimmer, R. Venkatesan, Y. S. Shao, J. Clemons, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. S. Emer, C. T. Gray, S. W. Keckler, and B. Khailany. A 0.11 pj/op, 0.32-128 tops, scalable multi-chip-module-based deep neural network accelerator with ground-reference signaling in 16nm. In *2019 Symposium on VLSI Circuits*, 2019.