AxE: An Approximate–Exact Multi-Processor System-on-Chip Platform

A. S. Baroughi*, S. Huemer†, H. S. Shahhoseini*, and N. TaheriNejad†
* Iran University of Science and Technology, Tehran, Iran
sadighbaroughi_a@elec.iust.ac.ir, shahhoseini@iust.ac.ir
† Technische Universitat Wien, Vienna, Austria
nima.taherinejad@tuwien.ac.at, e1027799@student.tuwien.ac.at

***Abstract*—Due to the ever-increasing complexity of computing tasks, emerging computing paradigms that increase efficiency, such as approximate computing, are gaining momentum. However, so far, the majority of proposed solutions for hardware-based approximation have been application-specific and/or limited to smaller units of the computing system and require engineering effort for integration into the rest of the system. In this paper, we present Approximate and Exact Multi-Processor system-on-chip (AxE) platform. AxE is the first general-purpose approximate Multi-Processor System-on-Chip (MPSoC). AxE is a heterogeneous RISC-V platform with exact and approximate cores that allows exploring hardware approximation for any application and using software instructions. Using the full capacity of an entire MPSoC, especially a heterogeneous one such as AxE, is an increasingly challenging problem. Therefore, we also propose a task mapping method for running exact and approximable applications on AxE. That is a mixed task mapping, in which applications are viewed as a set of tasks that can be run independently on different processors with different capabilities (exact or approximate). We evaluated our proposed method on AxE and reached a 32% average execution speed-up and 21% energy consumption saving with an average of 99.3% accuracy on three mixed workloads. We also ran a sample image processing application, namely gray-scale filter, on AxE and will present its results.**

***Index Terms*—Approximation Computing, Multi-Processor System-on-Chip (MPSoC), Approximate and Exact MPSoC, Task Mapping, RISC-V**

I. Introduction

Over the last decade, approximate computing as an emerging design paradigm has enhanced the efficiency of many applications such as image processing, multimedia, machine learning, and scientific computing, which can tolerate the occurrence of some errors. Exploiting approximate computing on RISC-V framework has been investigated lately, since RISC-V is a free and open Instruction Set Architecture (ISA), enabling a new era of processor innovation through an open standard. The energy efficiency of the present RISC-V cores shows that they are suitable for resource-constrained applications [1]. We advance the RISC-V based approximate computing field by creating an approximate and exact MPSoC together as a whole system. Due to the ever-increasing complexity of the computing tasks and systems, using the full capacity of the entire MPSoC, especially a heterogeneous one, becomes an increasingly challenging problem [2], [3]. However, having heterogeneous hardware and task-specific cores and accelerators (such as approximate accelerators) is necessary to perform efficiently.

In this paper, we present a heterogeneous MPSoC that benefits from both approximate and exact cores. We also propose -to the best of our knowledge, for the first time- a task mapping mechanism for it. In particular, we propose a task mapping for a system with approximate and exact computing cores. In our proposed method, applications are viewed as a set of tasks that can be run independently on different processors. Some can be approximated, and some cannot be (e.g., encryption tasks), and the mapper must find an optimum solution for assigning a heterogeneous task set to heterogeneous hardware. Our contributions are as following:

- Design and development of Approximate and Exact Multi-Processor system-on-chip (AxE): a Mixed-Approximate-Exact MPSoC framework based on RISC-V ISA, which benefits from approximate and exact cores,
- A mixed task mapping for exact and approximate tasks or programs on AxE, and
- Evaluating the impact of the proposed hardware and task mapping solution on various workloads and applications.

The remaining of this paper is organized as follows. In Section II, we first briefly review the respective literature and most related works. We describe the hardware and operation principles of our proposed AxE in Section III. We propose our method for mixed task mapper in Section IV. Experimental results and evaluations are described in Section V including an image processing showcase study. We present performance evaluation and comparison in Section VI, and finally, in Section VII, we conclude our paper.

II. Related Works

Exploiting approximate computing has been investigated lately. Respective efforts can be divided mainly into software and hardware-based approximation techniques. There are three commonly used software-based approximate computing techniques: loop perforation,input memorization, and output memorization [4]–[8]. The Loop perforation technique portrays skipping iterations, which may have a few types of perforation [5]. Input memorization techniques store the outcome of function calls to return the same results should the same inputs occur again. Some approaches relax the strict quality of inputs and devise approximate memorization [6].

Some may define the value similarity of the close-in-time inputs and forward the most comparable result [7]. Output memorization techniques assume that functions exhibit temporal output locality; thus, successive requests of the same function manage to deliver similar results [8]. Hardware-based approximations are mostly based on circuits that introduce errors in the outputs and are mostly used for error-tolerant applications such as machine learning and image processing. Since adders and multipliers are the basic computational units, approximate adders and multipliers have been investigated lately [9]–[11]. Furthermore, many applications perform extensive addition or multiplications on large amounts of approximable data. Approximation on such applications would be significantly more efficient. Approximate adders are mostly carry-disregarding at some stages or carry-prediction scheme based [12], [13]. Recently, approximate circuits, e.g., adders and multipliers, are more often used to assure area, latency, and energy efficiency [14]–[16]. Other circuits that benefit from approximation are storage circuits, e.g., Static RAM (SRAM). Such circuits and approaches are suitable for applications that require the processing of large volumes of data to achieve energy efficiency and can tolerate some accuracy loss [17], [18].

Very recently, exploiting approximate computing on the RISC-V framework has been investigated. Software adaptation has been investigated, and an ISA extension has been proposed alongside a control mechanism for multi-level accuracy [19]. [20] introduced a non-intrusive approach, which does not need source code modification, for approximation at the low-level in assembly, which allows approximating virtually all kinds of executable binaries. On the one hand, software approximation methods are attainable in target scenarios. On the other hand, approximate hardware methods are regularly demonstrated and considered in isolation, employed on specific application-level configurations. In [21], the authors demonstrated an extension for a RISC-V architecture that coordinates various hardware-level approximation techniques. The authors aim to lower the gap between software and hardware approximation techniques. As briefly reviewed above, in the previous research, capabilities and benefits of an MPSoC with both exact and approximate cores have not been explored. Such an MPSoC can benefit from approximate computing alongside exact computation capabilities. Hence, in this paper, we explore approximation on AxE, a multi-core system with exact and approximate cores. We also propose a task mapping method for AxE. To the best of our knowledge, the AxE and the task mapping method that we propose for running mixed workloads on AxE are the first of their kind.

III. Approximate and Exact Multi-Processor System-on-Chip (AxE)

A. High-Level Hardware Architecture

The overall number of cores and the type of cores in AxE is configurable. In this paper, we generally consider an MPSoC system with a total of T node, in which M cores are enhanced with an approximate computing accelerator and N cores are typical processors with exact computing capabilities only. Mapping different tasks to these cores based on the characteristics of the task, run-time requirements, capabilities, and availability of the cores is one of our main research questions.

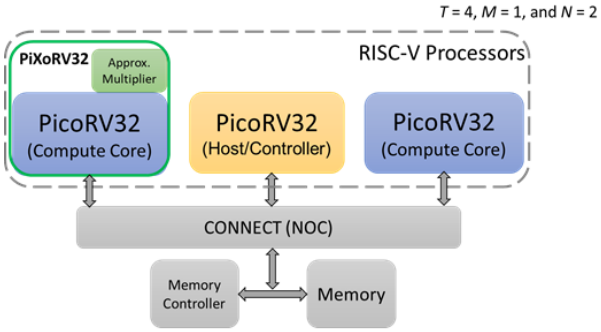

Fig. 1: Overview of the current instant of the proposed AxE framework.

For this paper, we have instantiated a specific version of this generic system, shown in Figure 1. Although both T and the ratio of M and N are configurable, for this paper, we chose $T = 4$, $M = 1$, and $N = 2$. One of the two typical cores is the host/controller processor and one node is the memory controller. This leaves one exact and one approximate core for the computation. The base cores are PicoRV32, which supports five RISC-V ISA extensions i.e., RV32E, RV32I, RV32IC, RV32IM, and RV32IMC. For our current instance, we chose RV32I and RV32IM subsets to keep the system as small as possible [22]. The approximate core is enhanced with an approximate multiplier called "EvoApprox8b", which we have obtained from [23]. In the rest of this paper, we refer to the PicoRV32 cores with approximate multipliers, as PiXoRV32. For the Network-on-Chip (NoC), we used a flexible RTL generator for fast, FPGA-friendly Networks-on-Chip (CONNECT) [24].

B. Approximate Multiplier Integration

The approximate core (PiXo) in AxE is based on the Pico core, utilizing an approximate multiplier. The Pico cores we have used implement the RISC-V RV32IMC Instruction Set. It can be configured as RV32E, RV32I, RV32IC, RV32IM, or RV32IMC core and optionally contain a built-in interrupt controller. The Pico core also utilizes a co-processor interface, which we use to add an approximate multiplier to Pico and obtain PiXo. The Pico Co-Processor Interface (PCPI) can implement non-branching instructions in external cores. When an unsupported instruction is encountered, and the PCPI feature is activated, then a validation signal is asserted, the instruction word itself is output on the instruction bus, the two sources fields are decoded, and the values in those registers are output on the respective PCPI outputs. An external PCPI core can then decode the instruction, execute it, and assert a ready signal when the execution of the instruction is finished. The PicoRV32 core will then decode the destination field of the instruction and write the value from PCPI output to the

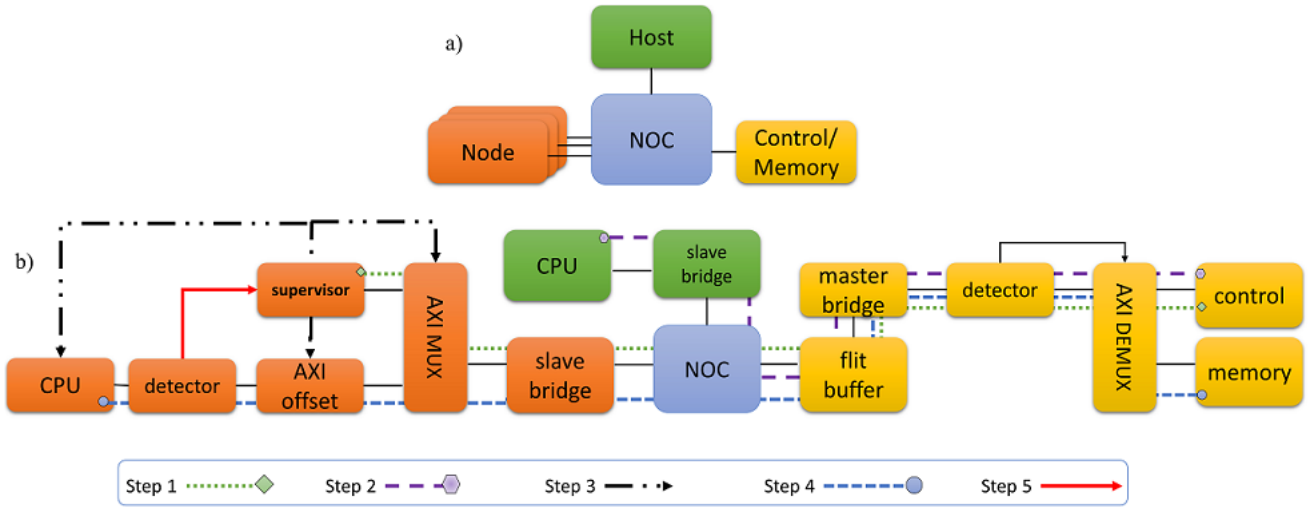

Fig. 2: AxE control scheme overview: a) system overview, b) control steps

respective register. When no external PCPI core acknowledges the instruction within 16 clock cycles, an illegal instruction exception is raised, and the respective interrupt handler is called.

C. System Operation

Since, in this work, applications are considered as a set of tasks that can be executed independently on different processors, the host node serves as a task mapper that points the processors to specified programs' address in memory and thus assigns them the respective task. The host turns on the processors whenever they have to run a task and turns them off when they have executed every task specified by the mapping program running on the host. This method is used under a clock gating scheme to ensure minimal power consumption and avoid system crashes due to "TRAP" signals.

D. Control Scheme

Our current instant of AxE consists of a host/controller node, two computational nodes, a memory controller, and an NoC. The control is done over the NoC and uses a control module to record what programs the nodes should execute. The control scheme can be described in 5 steps as follows (for a better understanding, we refer the reader to Figure 2 for visualization of the system and control scheme):

- **Node reading a program**: Once the system starts, the supervisor modules of the nodes initiate communication with the host to see if there are any program their respective Central Processing Unit (CPU) should execute. This is done via an Advanced eXtensible Interface Bus (AXI) read request. The programs are represented by their starting address in the memory and returned to the supervisor module in the *rdata* field of instruction. If *rdata* is 0, there is no program set for the node. Address 0 is reserved and is the start of the host/controller program, so no other program can start at this address. The read request from the supervisor module is special because of the address that is used. Every read and write request, where the Most Significant Bit (MSB) of the address is set, is considered a control signal. In the memory controller, the detector identifies these requests by checking the address and forwards these requests to the control by setting the AXI demultiplexer output (see Figure 2). Every other request is sent to the memory.
- **Host/Controller setting a program**: Via the software running on the host/controller CPU, a program can be set for a node, e.g., *NODE_1 = PROG_MUL;* The controller software uses #defines. For the previous assignment, the following is necessary:

```
#define PROG_MUL (0x1f0e8)
#define NODE_1_ADDR (0x80000098)
#define NODE_1 *((volatile int*)
↪ NODE_1_ADDR)
```

Here, 0x1f0e8 is a hex value that is the starting address of the corresponding program in the memory. So if a CPU started to read from this address, it would start to execute the program, for example, "PROG_MUL." The address used to identify the node in the control module is 0x80000098. This is the address that is used in the AXI requests. Generally, MSB=1 is specified for the control unit and not for the memory. When the address has an MSB=0, then it can access the memory. On the hardware side, the code for the control module can be summarized as follows:

```
pico_sel = latched_awaddr[
↪ PICO_MSB:PICO_LSB ];
if ( latched_awaddr[ INDEX_PROG ]==1'b1 )
begin
axi_offsets[ pico_sel ] = latched_wdata;
if ( latched_wdata == 0 )
    active[ pico_sel ] = 1'b0;
else
    active[ pico_sel ] = 1'b1;
end
```

- **Node reading a program**: Once the supervisor module reads an address that is not 0, it turns on the CPU and sets the AXI to offset the read address. This offset is needed

TABLE I: Three studied workloads

Programs	Workload 1	Workload 2	Workload 3
AES		*	*
blowfish	*	*	
dhrystone	*		
msort	*		
SHA256	*		
SpMVM	*	***	****

as every CPU starts reading from address 0 and each program is compiled the same way. The offset added to the address essentially moves all the reads and writes to the right memory space. So instead of reading from 0, the CPU reads from 0+0x1f0e8 in the case of the previous example. The same signal is used to turn on the CPU is also connected to the AXI multiplexer and switches the AXI communication from the supervisor module to the CPU. The supervisor module stops reading from the control and waits for the CPU to finish. In the hardware, such a read operation can be processed in the control module using:

```
pico_sel =  latched_araddr[
↪  PICO_MSB:PICO_LSB ];
if ( latched_araddr[ INDEX_PROG ]==1'b1 )
    latched_rdata = axi_offsets[pico_sel];
```

- **Node's CPU executing the program**: The node's CPU is now executing the program, reading from the memory as the MSB of the addresses is not set to 1, instead of reading from the control. Although the CPU of a node could read from the control unit, it is not intended to do so. While the CPU is executing the program, the host/controller can read from the control unit to get the state of the nodes. This operation returns the busy flag register. There is no way for the controller to interfere with the execution or terminate it. The busy flag register can be obtained by:

```
int busy = GET_BUSY;
```

- **Node's CPU signaling completion**: Once the execution on the node's CPU concludes, it writes a certain value to a certain address. This is detected by the detector who signals the completion to the supervisor module. The supervisor modules turn the CPU off and writes a 0 to the control, representing that there is now no program set for this node. As soon as this information has been received, the control also changes the flag in the busy register. After this step the *clock gating* module stops the clock on the node's CPU. Node's CPU remains off until the node's supervisor module finds new programs assigned to its respective CPU (Step 1).

IV. Task mapping on AxE

A. Mixed Task mapping

Since the PiXoRV32 has an exact multiplication execution unit alongside the approximate multiplication unit, it can execute both approximable and non-approximable tasks. On the other hand, PicoRV32 can execute only exact tasks, i.e., assigning approximable tasks to this core leads to exact calculations for those tasks. Therefore, to ensure the highest possible time savings, the task mapper has to ensure that approximable programs are assigned to the PiXoRV32 processor as far as possible and non-approximable ones are not unnecessarily assigned to it (especially, if PicoRV32 is free and available to run non-approximable tasks). Hence, we first consider assigning approximable tasks to the PiXoRV32 core and non-approximable tasks to the PicoRV32 queue. In some cases, this leads to an imbalance in the length of the two queues in terms of the necessary time to finish processing all tasks in the queue. Therefore, we move tasks from the lengthy queue to the slighter queue until they become equally protracted or switch place in the duration rank. If they switch places, we reconsider the assignment of the very last task that we replaced from one queue to another. Since the queues will not be equally long, we assign the reconsidered task to the PicoRV32 to ensure that if any of the two queues must be additionally succinct, it is the PiXoRV32 core that finalizes the execution of all assigned tasks earlier. This way, if any new approximable task has to be assigned, it has a shorter waiting time and, more importantly, does not have to be assigned to the PicoRV32 for exact computation, even though it could be approximated.

B. Workload Description

For our evaluations, we designed three workloads, shown in Table I, out of five non-approximable programs and one approximable program called Sparse-Matrix Vector Multiplication (SpMVM). The combinations of programs have been chosen to fairly create different load balances among non-approximable and approximable programs. The AES, blowfish, and SHA256 programs are related to encryption and cryptography; hence any approximation would conclude an undesirable result. The others do not have any calculations that PiXoRV32 can accelerate via approximation, i.e., they have no approximable multiplications. Each star in Table I shows the execution of one instant of that program; hence, more stars in a column show the repetition of that program in that workload. We note that these workloads serve as a proof-of-concept, and we are aware that for more thorough evaluations, we need a more extensive and diverse set of applications, which is among our plans.

V. Experimental Results

A. Speed

We ran the three workloads on the described hardware and compared it with the execution of the same workload using only exact cores. In other words, the tasks were mapped either on two exact computation cores, or on one exact and one approximate core (AxE). Table II presents the number clock cycles for execution of each studied program on fully exact and on our AxE systems at 50MHz clock frequency.

TABLE II: Execution cycles and energy consumption of studied programs

Programs	Execution Clock Cycles (millions)		Energy Consumption (pJ)	
	Exact	AxE	Exact	AxE
AES	82.58	82.58	8.38	8.67
blowfish	19.82	19.87	2.01	2.09
dhrystone	1.33	1.33	0.14	0.15
msort	1.07	1.07	0.11	0.12
SHA256	2.53	2.53	0.26	0.27
SpMVM	22.01	9.71	2.23	1.02

TABLE III: Power consumption details of the exact and approximate cores

	PicoRV32	PiXoRV32
Cells	6659	7258
Leakage Power (mW)	0.747	0.809
Dynamic Power (mW)	4.32	4.44
Avg. Power Consumption (mW)	5.07	5.25

B. Energy

To demonstrate energy consumption in our proposed method, we analyzed both PicoRV32 and PiXoRV32 cores in Cadence® Genus™ Synthesis Solution using 45nm NanGate technology. The power consumption and the area (number of cells) of each core are detailed in Table III. Based on these numbers, the energy consumption of each workload has been analyzed. The PiXoRV32 core is a bit larger than the PicoRV32 (it has both exact and approximate multipliers); hence, it has a slightly higher power consumption. Based on our evaluations, shown in Table III, the difference in the leakage power of the cores is negligible in comparison to the total power consumption (1%), which can be alleviated by using a *clock gating* scheme, as we mentioned in Section III-C. We explained in Section III-D how we integrated the *clock gating* scheme into the control scheme of AxE. Table II presents the energy consumption of each studied program on fully exact and on our AxE systems. The total energy consumption of a workload execution, E_t is calculated using:

$$E_t = \sum T_p \overline{P_p} + \sum T_x \overline{P_x} \tag{1}$$

where T_i, and $\overline{P_i}$ are the total (program) execution time, and average power consumption of the i core, respectively. $i \subset \{p, x\}$, where p represents the exact (PicoRV32) core and x represents the approximate (PiXoRV32) core. In our evaluations, the clock frequency for both exact and approximate cores is 50MHz and P_is are listed in Table III.

C. Accuracy

Since we have an approximable program, we evaluate the accuracy of the approximate program, which is present in all three studied workloads. For better illustration we report accuracy in percentages and as a complement of Mean Absolute Percentage Error (MAPE):

$$\text{Accuracy } (\%) = 100 * (1 - \text{MAPE}). \tag{2}$$

TABLE IV: Improvements in the image processing application

Input Image	Speed-up gain	Energy Improvement	Accuracy[1]
peppers	32%	27%	94.63%
fruits	32%	27%	93.55%
baboon	32%	27%	93.27%
lenna	32%	27%	95.30%

[1] Calculated by the definition in Section V-C and Equation (2).

TABLE V: Signal to Noise Ratio (SNR) and Peak Signal to Noise Ratio (PSNR) of the the Exact (Ex) and approximate (Ax) grayscale filters.

Input	SNR(dB)		PSNR(dB)		
	Ax	Ex	Integer to floating-point		Ax to Ex
			Ax	Ex	
peppers	7.77	7.90	21.90	22.40	30.99
fruits	9.80	9.90	20.27	20.39	28.99
baboon	9.83	9.95	18.74	20.01	30.25
lenna	9.27	9.40	19.47	20.65	30.04

where MAPE has been investigated for regression models, and Machine Learning (ML) applications [25], we calculate MAPE with respect to the definition as:

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^{n} \frac{|R_{p,i} - R_{x,i}|}{R_{p,i}} \tag{3}$$

where $R_{p,i}$ and $R_{x,i}$ are the i^{th} exact and approximate outputs (results), respectively, and n is the total number of all possible outputs.Therefore, the accuracy metric in this paper is calculated in percentages.

D. Application to Image Processing

In order to show the capacity of AxE for real-world applications, we used it for a sample image processing application, namely gray-scale filter that we applied to four standard 8-bit images, each has 128 pixels by 128 pixels. The output of a grayscale filter on AxE is illustrated in Figure 3. We present the speed-up and energy consumption gain in Table IV, and the SNR of the images in Table V.

As previously mentioned, the PicoRV32 and PiXoRV32 are using RV32IM subset of RISC-V ISA, in other words, these cores are integer-based (do not have floating-point units). Using integer-based cores for grayscale filter introduces some noise to the calculations compared to the floating-point filtering. The grayscaled images using exact and approximate cores share these noises in the minority of their pixels, but as we see on Figure 3, none of them are distorted aggressively. As expected, the grayscaled images filtered using the approximate core seem visually acceptable and intact. The PSNRs of the approximate filter to the exact filters shown in Table V are in all cases except one above 30dB, which speaks to the acceptable quality of the conversion using the proposed approximate core. The only case with lower PSNR is the fruits image, which has a PSNR very close to 30dB (28.99dB). Even though minor noises can be found upon a close look at the image, they are not significant, especially compared to the significant energy and speed-up gains achieved. Therefore, we

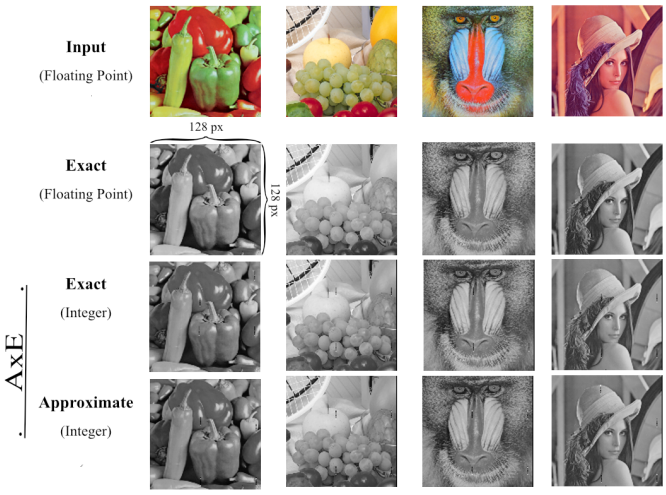

Fig. 3: Grayscale filter on AxE and other platforms.

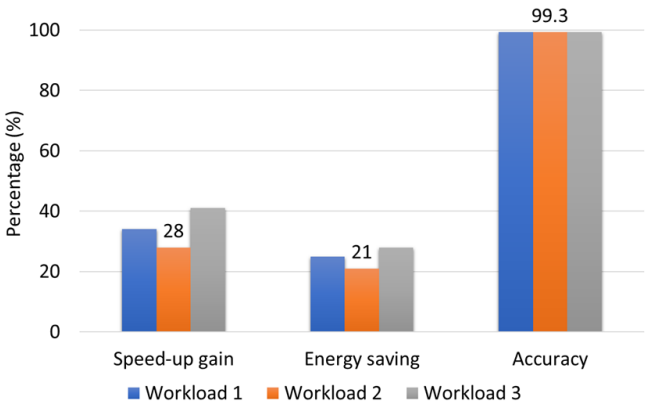

Fig. 4: Energy improvement and execution speed-up of three workloads

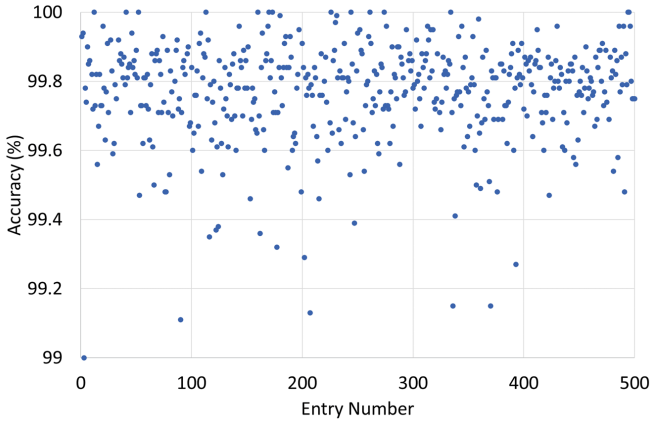

Fig. 5: Accuracy of approximate program for each output entry

contend that the proposed system is an acceptable solution for the application at hand.

VI. Performance Evaluation and Comparison

A. Speed

We executed the three workloads on the described hardware and compared it with executing the same workload using only exact cores. In other words, the tasks were mapped either on two exact computation cores or one exact and one approximate core (AxE). In Figure 4, we show the speed-up gain in each application. That is, the number of clock cycles to finish the workload using the proposed mixed-approximate-exact (AxE) system normalized to that of the fully exact system. Our evaluation of three workloads shows that on average, our approximate and exact (AxE) system has 1.32x speed of the fully exact system (i.e., on average 32% faster).

B. Energy

The power consumption and the area (number of cells) of each core are detailed in Table III. Based on these numbers, the energy consumption of each workload has been analyzed by Equation (1). The energy consumption saving using our proposed solution compared with an exact MPSoC shown in Figure 4. We consider the difference in energy consumption is mainly in cores, which is much more than the energy difference in other parts of our system. In addition, their energy consumption would be the same in both cases of fully exact and mixed-approximate-exact (AxE) MPSoCs. As visualized in Figure 4, the energy-saving in comparison to an exact MPSoC is at least 21% (for the second workload) and on average 25%.

C. Accuracy

Since we have an approximable program, we evaluate the accuracy of the approximate program, which is present in all three studied workloads. The SpMVM program contains 2500 multiplication and results in a vector with 500 indices. By integrating the approximate multiplier on PiXoRV32, running the approximable program in each workload on our mixed-approximate-exact MPSoC introduces some errors in the multiplications. We obtained 99.3% average accuracy (MAPE equals 0.7%) on the approximable program driven from Equation (2) and Equation (3). The actual accuracy of each output indices and the distribution of the approximate output vector is shown in Figure 5.

VII. Conclusion

In this paper, we presented a Mixed-Approximate-Exact MPSoC framework called AxE and a preliminary solution for the challenge of task mapping for such a system. Our evaluations showed that the studied instance of AxE and the proposed task mapper executed on our framework led to a considerable (32%) execution speed-up compared to an exact-only MPSoC. The energy-saving is 21% while achieving 99.3% average accuracy on approximate program among three studied workloads. Our image processing showcase study showed that the proposed system has an acceptable quality while maintaining a 32% speed-up gain and 27% energy-saving on average. This framework provides a basis for further

works in this field. Studying instances with a more significant number of nodes and heterogeneity is a natural extension of this work. Besides a more extensive system, we plan to verify our preliminary results with a more extensive set of workloads with more variations. Adding other approximate computing units and studying their effect is another extension of this work, which we plan to pursue in the future.

REFERENCES

[1] I. Elsadek and E. Y. Tawfik, "RISC-V resource-constrained cores: A survey and energy comparison," in *2021 19th IEEE International New Circuits and Systems Conference (NEWCAS)*, 2021, pp. 1–5.

[2] E. Shamsa, A. Kanduri, N. TaheriNejad, A. Proebstl, Chakraborty, A. M. Rahmani, and P. Liljeberg, "User-centric resource management for embedded multi-core processors," in *The 33rd International Conference on VLSI Design and The 19th International Conference on Embedded Design*, 2020, pp. 1–6.

[3] E. Shamsa, A. Proebstl, N. TaheriNejad, A. Kanduri, A. M. Rahmani, P. Liljeberg, and Chakraborty, "Ubar: User and battery aware resource management for smartphones," *ACM Transactions on Embedded Computing Systems (TECS)*, pp. 1–23, 2021.

[4] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 124–134.

[5] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard, "Quality of service profiling," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1, 2010, pp. 25–34.

[6] A. K. Mishra, R. Barik, and S. Paul, "iACT: A software-hardware framework for understanding the scope of approximate computing," 2014.

[7] Y. Kim, S. Venkataramani, S. Sen, and A. Raghunathan, "Value similarity extensions for approximate computing in general-purpose processors," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 481–486.

[8] G. Tziantzioulis, N. Hardavellas, and S. Campanoni, "Temporal approximate function memorization," *IEEE Micro*, vol. 38, no. 4, pp. 60–70, 2018.

[9] J. Lee, H. Seo, H. Seok, and Y. Kim, "A novel approximate adder design using error reduced carry prediction and constant truncation," *IEEE Access*, vol. 9, pp. 119 939–119 953, 2021.

[10] S. E. Fatemieh, M. R. Reshadinezhad, and N. TaheriNejad, "Approximate in-memory computing using memristive imply logic and its application to image processing," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022, pp. 1–5.

[11] P. J. Edavoor, S. Raveendran, and A. D. Rahulkar, "Approximate multiplier design using novel dual-stage 4:2 compressors," *IEEE Access*, vol. 8, pp. 48 337–48 351, 2020.

[12] G. Anusha and P. Deepa, "Design of approximate adders and multipliers for error tolerant image processing," *Microprocessors and Microsystems*, vol. 72, p. 102940, 2020.

[13] H. Seok, H. Seo, J. Lee, and Y. Kim, "COREA: Delay- and energy-efficient approximate adder using effective carry speculation," *Electronics*, vol. 10, no. 18, 2021.

[14] S. Ullah, S. Rehman, B. S. Prabakaran, F. Kriebel, M. A. Hanif, M. Shafique, and A. Kumar, "Area-optimized low-latency approximate multipliers for FPGA-based hardware accelerators," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: Association for Computing Machinery, 2018.

[15] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "TOSAM: An energy-efficient truncation- and rounding-based scalable approximate multiplier," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 5, pp. 1161–1173, 2019.

[16] R. Nayar, P. Balasubramanian, and D. L. Maskell, "Hardware optimized approximate adder with normal error distribution," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. Los Alamitos, CA, USA: IEEE Computer Society, July 2020, pp. 84–89.

[17] Y. Chen, X. Yang, F. Qiao, J. Han, Q. Wei, and H. Yang, "A multi-accuracy-level approximate memory architecture based on data significance analysis," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 385–390.

[18] M. Kang, S. K. Gonugondla, and N. R. Shanbhag, "Deep in-memory architectures in SRAM: An analog approach to approximate computing," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2251–2275, 2020.

[19] T. Trevisan J *et al.*, "Approxrisc: An approximate computing infrastructure for RISC-V," RISC-V Workshop in Barcelona, May 2018, poster.

[20] N. A. Said *et al.*, "FPU bit-width optimization for approximate computing: A non-intrusive approach," in *2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2020, pp. 1–6.

[21] I. Felzmann, J. F. Filho, and L. Wanner, "Risk-5: Controlled approximations for RISC-V," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4052–4063, 2020.

[22] C. Wolf, "PicoRV32 - A Size-Optimized RISC-V CPU." [Online]. Available: https://github.com/cliffordwolf/picorv32

[23] V. Mrazek *et al.*, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *DATE, 2017*, 2017, pp. 258–261.

[24] M. K. Papamichael and J. C. Hoe, "Connect: Re-examining conventional wisdom for designing NOCs in the context of FPGAs." Association for Computing Machinery, 2012.

[25] A. De Myttenaere, B. Golden, B. Le Grand, and F. Rossi, "Mean absolute percentage error for regression models," *Neurocomputing*, vol. 192, pp. 38–48, 2016.