# Approximation-aware Task Partitioning on an Approximate-Exact MPSoC (AxE)

S. Huemer[†], A. S. Baroughi[*], H. S. Shahhoseini[*], and N. TaheriNejad[§†]
[†] Technische Universitat Wien, Vienna, Austria
stefan@huemer.tech
[*] Iran University of Science and Technology, Tehran, Iran
sadighbaroughi_a@elec.iust.ac.ir, shahhoseini@iust.ac.ir
[§] Institute of Computer Engineering (ZITI), Heidelberg University, Heidelberg, Germany
nima.taherinejad.ziti.uni-heidelberg.de

***Abstract*—As the demand for increased performance and reduced energy consumption continues to grow, Quality of Service (QoS) adjustment approaches offer an effective way to tackle those demands. One such method, approximation, has gained popularity in recent years, facilitating faster executions as well as a smaller power consumption by providing an approximated result. The areas in which these trade-offs are acceptable are numerous, but hardware-based solutions are usually domain-specific and expensive to integrate. To tackle this issue, we take a different approach, in which approximate hardware can be used (or not) in a general purpose environment and via software decisions. That is, a Multi-Processor System-on-Chip (MPSoC) that contains Central Processing Units (CPUs) that offer approximate calculations alongside the ones that offer exact calculations. However, current task partitioning algorithms do not consider the specific capabilities or requirements of such a MPSoC. This paper introduces approximation-aware partitioning algorithms using different strategies and compares the results to the State-of-the-Art (SoA). Additionally, the resulted task partitions are executed to gauge their quality compared to the SoA. Experimental results show, that the usage of an approximate CPU and approximation-aware task partitioning leads to an increased partition success rate of $21.5\%$. Furthermore, the execution, i.e., scheduling of the partitioned tasks until energy starvation, achieves a $3.4\%$ extended run-time.**

***Index Terms*—Approximate-Exact MPSoC, task partitioning, energy harvesting, RISC-V.**

## I. Introduction

The efficiency of several applications, including image processing, multimedia processing, machine learning, and scientific computing, which can tolerate a certain level of inaccuracy, has increased over the last ten years as a result of approximate computing as an emerging design paradigm [?], [1]–[7]. Since RISC-V is a free and open instruction set architecture (ISA), allowing a new age of processor innovation via an open standard, exploiting approximation computing on RISC-V framework has been studied recently. The energy efficiency of the current RISC-V cores demonstrates that they are appropriate for applications with limited resources [8], [9].

While approximate computing has gained a lot of attention as a method to reduce Quality of Service (QoS) for higher performance, utilizing approximation in a heterogeneous Multi-Processor System on a Chip (MPSoC) consisting of approximate as well as exact calculating Central Processing Units (CPUs) allows a system to benefit from approximation as well as exact calculations. However, existing partitioning algorithms do not take the special characteristics of mentioned system into account, thus creating partitions that might be invalid. Not considering the benefits of approximation might cause the resulting partition to be sub-optimal, but does not result in an invalid partition. However, should a task that requires exact computation to reach a correct result, be assigned to an approximate CPU, the partition is invalid as the result is incorrect. This is mostly the case in cryptography tasks, or calculated memory accesses.

To investigate potential improvements from such a system, we consider a task-set of independent non-preemptive periodic tasks having firm deadlines. This task-set is partitioned on a MPSoC, called Approximate-Exact MPSoC (AxE), comprising 2 nodes, one having an approximate and the other one an exact CPU, the difference as the former one approximates multiplications. The partitioning is done by a controller node which can schedule the partitioned tasks as well. The communication between the controller and the other nodes, as well as the memory access of the latter, is done via an Network on a chip (NoC).

The contributions of this paper include:

- Approximation-aware task partition algorithms.
- Evaluation and comparison of the performance and success rate of different approximation-aware strategies.
- Scheduling of the partitioned tasks to investigate the quality of the partitions, i.e., how long they can run until energy starvation sets in.

The rest of this paper is organized as follows: In Section II, we review related work briefly. The MPSoC used, is described in Section III, and Section IV defines the models. Section V explains the proposed approximation-aware task partitioning algorithms and their characteristics. Section VI briefly explains the scheduling algorithm used on the task partitions. Section VII shows and discusses the results, and finally Section VIII concludes the paper.

979-8-3503-3757-0/23/$31.00 ©2023 IEEE

## II. Related Works

The partitioning algorithm that is used as a basis and a reference, Energy Harvesting - Reasonable Allocation (EH-RA) was introduced in [10] as a bin-packing approach used on a Multi-Processor System-on-Chip (MPSoC) utilizing energy harvesting. In [11] the scheduling algorithm, Earliest Deadline - Harvesting (ED-H) was presented which was also utilized in conjunction with an energy harvesting system.

Reference [12] proposed an Approximate-Exact MPSoC (AxE) system, which is also used in this research. Additionally the special considerations of such a system are discussed, i.e., the need for a task requiring exact calculations to a CPU providing exact calculations. Reference [13] proposed another general purpose system offering approximation techniques. They introduced Risk-5, a RISC-V extension, offering a low-overhead software interface to enable or disable approximation functionality via Control and Status Registers (CSRs). It is noteworthy that the authors offer the possibility of deactivating any hardware modules that are not in use to further reduce the energy consumption. ISA extension and multi-level precision control mechanisms have been researched for software adaptability in [14] and reference [15] proposed a non-intrusive assembly technique for approximating thus allowing any software to make use of it without any change to the source code.

Approximation was used during the task partitioning in [16] and [17]. The approximation approach is used to find a solution that maximizes the number of tasks that are successfully allocated. However, the MPSoC which used in that study does not support approximate computing.

To our knowledge, no prior research has adapted task partitioning for use in a MPSoC consisting of both exact as well as approximate CPUs.

## III. Approximate and Exact Multi-Processor system-on-chip (AxE)

Figure 1 illustrates an overview of the system that is used to conduct the research. The system contains 2 nodes executing tasks, an NoC based on CONfigurable NEtwork Creation Tool (CONNECT) [18], a memory controller with a Block Random Access Memory (BRAM) and a controller in charge of partitioning and scheduling tasks. The controller is also connected to a Universal asynchronous receiver-transmitter (UART) facilitating the transmission of information regarding the system. It is a heterogeneous system where the CPU of one node uses exact multiplication while the CPU of the other one uses approximate multiplication. This system can be explored on FPGA platform, but in this paper we present the software emulation of this system, where reported hardware-related numbers are obtained through Application Specific Integrated Circuit (ASIC) synthesis on a 45nm technology. As a reference, a version of this system is used, referred to as ExE that contains only nodes with exact multiplication.

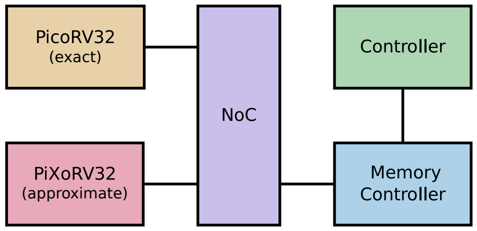

Fig. 1: Overview of AxE system.

### A. CPUs

The CPUs of the nodes are instances of the PicoRV32 [19], a RISC-V based CPU.

It was chosen due to its small size and the co-processor interface it supports, the Pico Co-Processor Interface (PCPI), an interface to extend the functionality of the CPU utilizing custom non-branching instructions.

*1) PicoRV32:* The PicoRV32 is an instance to use the RV32IM instruction set.

*2) PiXoRV32:* Based on the PicoRV32, the PiXoRV32 has been created by incorporating approximate multiplication [12]. The approximation circuit has been obtained from the EvoApproxLib project [20] first presented in [21]. In this instance, the multiplier mul16s_GV3_pdk45, a signed 16-bit multiplier, has been used. This is done either as a replacement to the exact multiplication, or as an additional instruction utilizing the PCPI. For this research the latter option was chosen, i.e., replacing the exact multiplication, to keep the size as small as possible.

## IV. Models and Definitions

We explain the characteristics of nodes and the properties of tasks, including their worst-case execution time, energy requirement, and relative deadline. Additionally, we present the task-set used in the experiments, including the improvement in execution time on an approximate CPU, if applicable.

### A. Nodes

Each node $P_j$ is characterized by:

- $E_j$: the charge available to the node
- $eMax_j$: the highest worst-case energy requirement of the tasks assigned to the node
- $arch_j$: the architecture of the node (exact or approximate)
- $\Gamma_j$: the tasks assigned to the node

The total power consumption of the different nodes, or more specifically, their respective CPUs has been analysed at 45nm NaNgate Technology [12]:

- PicoRV32, the exact CPU: $5.07mW$
- PiXoRV32, the approximate CPU: $4.76mW$

For simplicity a node containing a CPU using an exact or approximate multiplier will be referred to as an exact node and an approximate node, respectively.

### B. Tasks

Each task $\tau_i$ is periodic, non-preemptive and characterized by the following properties:

- $C_i$: worst-case execution time for each available architecture
- $E_i$: worst-case energy requirement for each available architecture
- $D_i$: relative deadline
- $T_i$: period

The worst-case execution time $C_i$ for each task has been determined by executing it on a node of the system while recording the start and end times using both to calculate the difference. Note that every task has to be executed on all architectures that are available i.e. once on an approximate node and once on an exact node. It should be mentioned, during this evaluation, the other nodes are not executing any tasks. However, the results are valid as there is no advantage for the node executing the task currently under investigation by running on a system which is otherwise idle. As the CPUs of the other nodes might be idle, the node is still polling the memory for tasks to execute. This causes traffic over the NoC similar to the traffic created by a CPUs executing task.

Using this worst-case execution time and the power measurements of the node, the worst-case energy requirement $E_i$ is calculated. Like the worst-case execution time, this is done for each available architecture.

The relative deadline $D_i$ as well as the period $T_i$ are not dependent on the architecture of the node and have the same value i.e., $0 < C_i < D_i = T_i$ for each task $\tau_i$. It is set to a random multiple of the worst-case execution time of the architecture providing the exact multiplication i.e., $D_i = T_i = [ran_1 : ran_2] * C_{i_{exact}}$ with $0 < ran_1 < ran_2$. In contrast to the worst-case execution time and the worst-case energy requirement, this value is the same regardless of the architecture of the node the task is assigned to. Practically this means that every task that executes faster using approximation and is assigned to an approximate node has more slack time then i.e., more time to execute the task before the deadline.

### C. Task-set

A task-set of $n$ independent periodic tasks is defined as $\Gamma = \tau_1, \tau_2, ..., \tau_n$.

For the experiments the task-set presented in Table I is used. This task-set is initially designed to explore the potential of employing such a system and serves as a proof-of-concept. In this table, the worst-case execution time is shown for each task and each architecture as well as the difference between them, i.e., the improvement if they are being executed on an approximate CPU. Should there be no difference between the two times, then the task simply has no multiplications and thus no instruction that is effected by the approximation.

## V. Approximation-aware Task Partitioning

As the worst-case execution times of a task are known for both approximate as well as exact nodes before they are partitioned, it would seem trivial to pick the ones that have a smaller execution time on an approximate node and prioritize the assignment to it. However, there are some tasks that should not be approximated even if they would have a better execution time and energy profile on an approximate node [12]. For example, a cryptography task becomes pointless if approximated and memory accesses to indices determined by approximation will lead to invalid memory accesses e.g. SHA-256. Table II shows the tasks and how they are effected by approximation. Examining the entry for ‘qsort’, we can see that we gain an improvement if it is executed on an approximate node. However, this would result in invalid memory accesses, meaning, that approximation is not applicable. Note that although ‘aes’ is a cryptography task, our implementation does not use multiplications and therefore could be executed on an approximate node with any repercussions.

For simplicity, tasks requiring exact multiplication are going to be referred to as exact tasks and tasks that benefit form approximation and do not require exact results, approximate

TABLE I: Task-set

| Program | Execution time $C_i$ in ms | | Difference in ms |
|---|---|---|---|
| | exact | approx | |
| aes | 1651.60 | 1651.60 | 0 |
| blowfish | 397.57 | 397.57 | 0 |
| dhrystone | 26.63 | 26.52 | 0.12 |
| grayscale | 89.83 | 25.55 | 64.28 |
| msort | 21.45 | 21.45 | 0 |
| norx | 57.09 | 57.09 | 0 |
| primes | 2.69 | 2.69 | 0 |
| qsort | 26.93 | 5.22 | 21.71 |
| sha256 | 50.50 | 50.44 | 0.06 |
| sharpen | 95.57 | 78.07 | 17.50 |
| square_mmult | 9.61 | 7.67 | 1.94 |

TABLE II: Approximability of the task-set

| Task | Approximation | | Improvement | Type |
|---|---|---|---|---|
| | applicable | NA[1] | | |
| aes | ✓ | | | neither |
| blowfish | ✓ | | | neither |
| dhrystone | ✓ | | ✓ | approx |
| grayscale | ✓ | | ✓ | approx |
| msort | ✓ | | | neither |
| norx | ✓ | | | neither |
| primes | ✓ | | | neither |
| qsort | | ✓ | ✓ | exact |
| sha256 | | ✓ | ✓ | exact |
| sharpen | ✓ | | ✓ | approx |
| square_mmult | ✓ | | ✓ | approx |

[1](Approximation is) Not Applicable.

tasks. There is a third class of takes that don't benefit from approximation and don't require exact execution, i.e., they contain no multiplication instructions, referred to as "neither" here.

The requirement of having exact tasks assigned to an exact node introduces an additional criteria for a partition to fail. In addition to the the processor utilization having to be smaller than 1 i.e., $U_p < 1$, the utilization specifically to the exact node must also be smaller than 1 i.e., $U_{p_{exact}} < 1$. The latter simply means, that if an exact task does not fit on an exact node, the criteria is violated and the partition invalid. Note that the energy-requirement is not taken into account here as it is proportional to the worst-case execution time.

As the preferred assignment for both approximate as well as exact tasks is clearly their corresponding node, the focus of the different strategies is on what to do with tasks that are neither exact nor approximate. The pseudo code for all strategies is shown in Listing 1. We present them in the rest of this section.

**Listing 1** Pseudo code of approximation-aware strategies.

```
def assign_tasks():
  for task in tasks:
    # pick candidate nodes
    for node in nodes_list:
      if ( node.util + task.util ) < 1:
        nodes.add( node )
    if nodes.is_empty():
        return partition_failed
    node = find_node( task, nodes )
    if node:
      node.assign( task )
      node.utilization += task.utilization
  else:
   return partition_failed

def find_node( task, available_nodes):
  # if the task is approximateable and
  # there is an approximate node in the
  # candidate nodes, then return the
  # first approximate node in the list
  if task is approx and nodes.have(approx):
    return nodes.get_approx()
  elif task is exact and nodes.have(exact):
    return nodes.get_exact()

  if strategy is AA-b:
    return nodes.get_emptiest()

  elif strategy is AA-a:
    if task is neither and nodes.have(exact):
      return nodes.get_exact()
    elif task is neither and nodes.have(approx):
      return nodes.get_approx()
    elif task is approx and nodes.have(exact):
      return nodes.get_exact()

  elif strategy is AA-e:

    if task is neither and nodes.have(approx):
      return nodes.get_approx()
    elif task is neither and nodes.have(exact):
      return nodes.get_exact()
    elif task is approx and nodes.have(exact):
      return nodes.get_exact()

  return partition_failed
```

### A. *Approximation-aware approximate focus (AA-a)*

To achieve the highest benefit of approximation, the most straightforward way is to make sure that every approximate tasks fits on an approximate node. In order to make sure these assignments can be done, tasks with no preference are assigned to an exact node. Therefore the load on the approximate nodes is kept low, allowing approximate tasks to be assigned to it, with a high probability.

### B. *Approximation-aware exact focus (AA-e)*

This safer strategy foregoes possible improvements of approximation to make sure, that exact tasks fit on an exact node. Other tasks are therefore assigned to an approximate node, i.e., the load of the exact nodes is kept as low as possible.

### C. *Approximation-aware balanced (AA-b)*

The last strategy uses the emptiest node, i.e., the node with the least processor utilization, for assignments of tasks with no preference. Although there is no special attention on exact tasks, it is expected that the overall load is small enough for an exact node to accommodate it.

## VI. Task Scheduling

A successful partition does not necessarily reflect the partition's quality in terms of executing the tasks with efficiency from the point of view of low energy consumption and balanced utilization on CPUs.

To compare the partitioning results, the algorithm ED-H [11] is used to schedule them. This algorithm was presented for use in a system utilizing energy harvesting and is chosen as it places additional restrictions on a system model, making its execution more challenging. In this paper, a battery model is used, which considers a separate battery connected to each node. As the tasks cannot be reassigned after the partition, the system fails if the battery of one node is depleted.

It follows the Earliest Deadline First (EDF) approach of picking the task with the earliest deadline first and additionally checks to see if the execution of the task causes energy starvation. Should this be the case, the execution is delayed until the related battery has enough charge. The energy storage is considered empty if the following applies: $0 \leq E(t) < eMax$, with $eMax$ being the highest worst-case energy requirement of the tasks in the task-set.

Therefore, even if there is enough charge left to execute a task, the system idles until that battery has recharged to at least $eMax$. Should that idle time be too long and a task misses its deadline, then the system is considered to be starved of energy.

## VII. Results and Discussion

### A. *Approximation-aware task partitioning*

*1) Setup:* In order to get variation into the task-set, the periods of the tasks are determined by a random variable multiplied with a worst-case execution time of the tasks.

This random number is determined by a random number generator that uses a range of 5 i.e., $[r : r+5]$. Beginning with the range $[1 : 6]$, the boundaries of the range are increased by

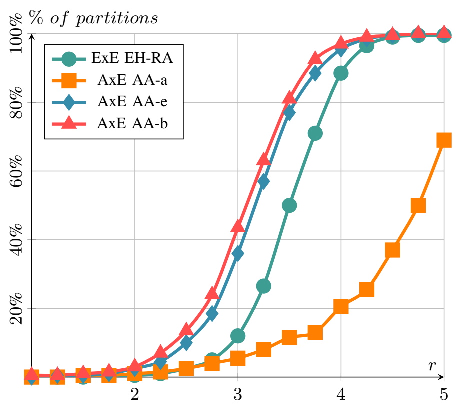

Fig. 2: Average partition success rate.

0.25 for each run until the range of $[5:10]$ is reached. For each step, the task-set is partitioned 100 times, defining the success rate of the algorithm in that range. Overall the success rate is expected to improve with the higher range as the task utilization will shrink and therefore putting a smaller load on the nodes they are assigned to.

*2) Results:* Figure 2 illustrates the average successful partitions using the proposed algorithms as well as State-of-the-Art (SoA) EH-RA algorithm. The latter will be used as a reference to see how much can be improved by approximation. Consider that we are running EH-RA on, ExE, a system having 2 exact nodes since this algorithms does not consider the level of heterogeneity involved in AxE. The proposed approximation-aware algorithms are running on, AxE, a system having an approximate node as well as an exact one.

For AA-a, successful partitions do start pretty early on in comparison with EH-RA, but it takes a long time for the best-case to surpass even $10\%$ in the range of $[3:8]$ and the average case achieving this at $[3.5:8.5]$. This is far worse than the ExE system configuration using EH-RA, voiding all the gains from approximation. Figure 3a shows that the criteria $U_{p_e} > 1$ is the main reason for the poor performance. Note that AA-a strategy favors the exact node for any task that has no clear preference, thus making it harder for exact tasks to be assigned specifically to an exact node.

The percentage of successful partitions using AA-e has greatly improved over the previous approach and is on average better than the success rate of the ExE system. Figure 3b shows that failures are mostly due to processor utilization violations, i.e., $U_p > 1$. Failures due to an exact task not fitting on an exact node have greatly diminished.

A minor improvement is gained by using AA-b which focuses on keeping the load of the nodes balanced. Figure 3c depicts an increased failure rate due to exact tasks not having enough space on an exact node, i.e., $U_{p_{exact}} > 1$. However, due to the overall reduced load obtained by AA-b, the failures due to $U_p > 1$ are reduced sufficiently, that this approach has a better partition success rate compared to the previous one.

### B. Task partition schedulability/longevity

To estimate the quality of the task partitions, we schedule the partitions using ED-H disregarding potential charges by the energy harvesters, thus running the system until one of the nodes starves of energy. The initial charge is set to $10,000J$ for each node.

Figure 4 shows the very last seconds before the energy starvation set in. As mentioned above, tasks cannot be reassigned to another node and that each node has its own battery. Thus, the execution of a partition can fail if the partitioning algorithm put a disproportional load on one of the nodes, causing the corresponding battery to drain quicker and the system starving of energy wasting the charges of the other batteries. This is the reason that the energy starvation can set in while the total remaining energy is still quite high. Most noticeable is this in the case of the AA-e strategy that failed with more than $450J$ charge left in the battery of the other node.

The scheduling based on the AA-a partitioning algorithm was successful despite AA-a's low success rate. However, if a task set can be partitioned with this method, the highest possible improvement form approximation can be achieved. This is due to the preferred assignment of tasks with no preference to an exact node, thus making sure any approximate task can be assigned to an approximate node. Therefore, this puts a disproportional load on the exact node and leads to starvation of energy.

Another reason this method is scheduled so successfully is the partition of the 'aes' task to the approximate node. This task has a disproportional worst-case execution time compared to the rest and, thus, also a disproportionate worst-case energy requirement. Therefore, it is hard to find a node that has a load small enough to accommodate this task. As the AA-a partitioning algorithm puts the most load to the exact node, which results the approximate one to stay relatively empty and the 'aes' task is assigned to it and not the exact one. In all other cases, the exact node is executing the 'aes' task. Simply speaking, the execution of any tasks on an approximate node requires less energy than on an exact one because the approximate one requires less power. Although there is no benefit of this task being assigned to the approximate node, the generally lower energy requirement causes the load of this task to not have such an impact on the overall performance of the system.

Utilizing AA-e yields a great partition success rate. However, its run-time is very poor, yet better than the reference system using 2 exact nodes. The reason for this is the small gain from approximation. As any task with no clear preference has been assigned to the approximate node, thus causing approximate tasks not to fit on it anymore.

Using the third and last approach, AA-b the load of the nodes is spread more evenly, resulting in a slight improvement over AA-a. Although the 'aes' task is assigned to the exact node and ultimately prompted the energy starvation, the load balancing made sure that not too much burden is placed on the exact node.

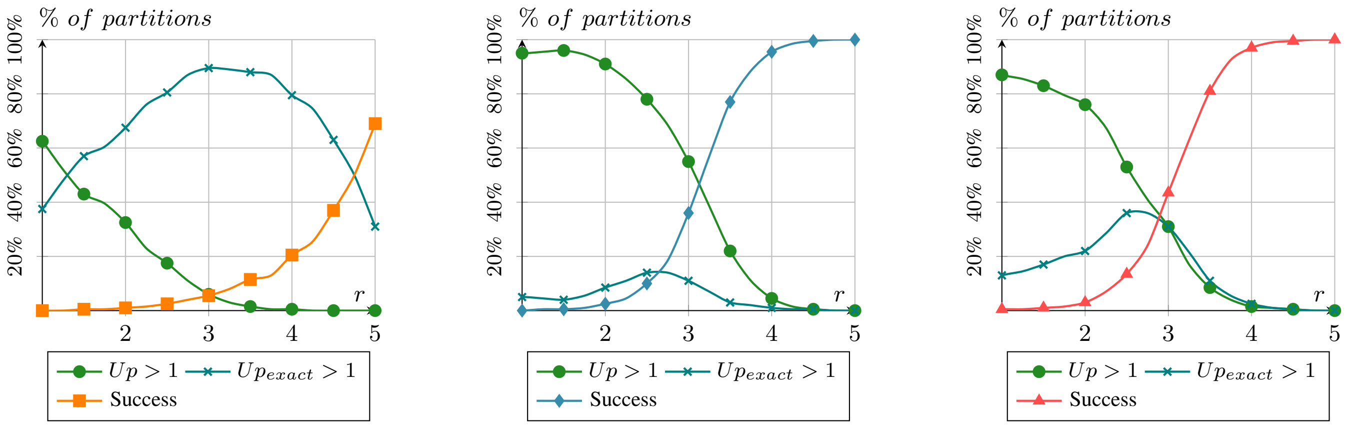

Fig. 3: Reason for partition failures for the proposed strategies (a) AA-a (b) AA-e (c) AA-b.

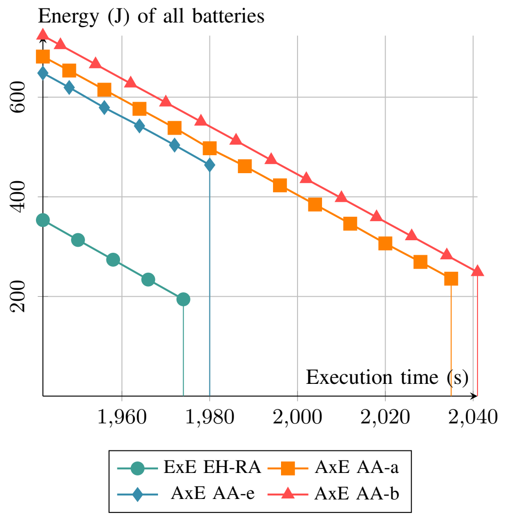

Fig. 4: Run time (s) until energy starvation

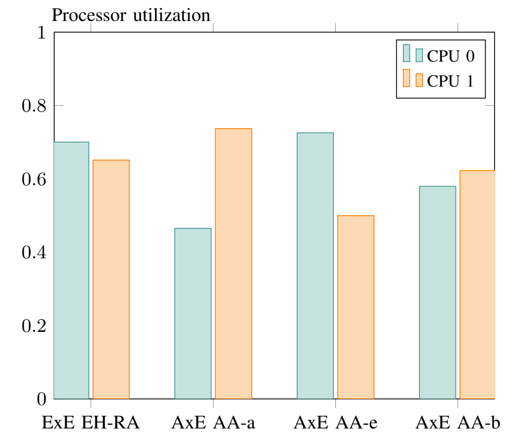

Fig. 5: Processor utilization for different partitioning algorithms

### C. Processor utilization

Figure 5 shows the processor utilization of the proposed algorithms as well as EH-RA. Although the AA-a and AA-e algorithms are imbalanced, AA-b was able to spread the load more evenly. By observing the utilization of EH-RA and AA-b you might notice, that the overall utilization of AA-b is lower. This is due to the approximation which lowers the task's utilization and thus reduces the process utilization.

## VIII. Conclusion

In this paper, the potential improvements that a software-programmable approximate-hardware in a general purpose MPSoC platform were studied. In particular, a MPSoC executing periodic tasks has been investigated. Although, traditional task partitioning algorithms are not made to deal with the special requirements of such a system, they can be adapted and yield significant improvements in performance. For the experiments, algorithms dealing with low-power energy-harvesting systems have been used as a basis for this research, since a major reason for using approximate computing is saving energy.

By increasing the period of the tasks we observe the partition success rates using the different approaches. The improvement was demonstrated very well when the period was set to a random multiple in the range $[3.75 : 8.75]$ of the worst-case execution time, i.e., $T_i = C_i * rand[3.75 : 8.75]$. On average the AxE configuration could be partitioned $92.5\%$ of the time using AA-b, while ExE using EH-RA only achieves $71\%$, i.e., an increase of $21.5\%$.

Based on these partitions the system was executed and the tasks scheduled using ED-H. In the experiments ExE, the reference system, using EH-RA for the task partition was only able to run for $1974s$, while AxE using AA-b achieved a run-time of $2041s$, an additional $67s$ or $3.4\%$. Given the nature of energy-harvesting system this means more than an additional minute of charging, potentially avoiding energy starvation by bridging a time frame of low energy generation. We note that in the tasks that more approximate multiplications can be used, or hardware that benefits from other approximate accelerators, this advantage is expected to scale up. In our future work, we plan to focus on task partitions on such MPSoC with more nodes and a more extensive and diverse task-set.

## References

[1] G. Burel, H. Saif, M. Fernandez, and H. Alani, "On semantics and deep learning for event detection in crisis situations," 2017.

[2] N. Amirafshar, A. S. B., H. S. Shahhoseini, and N. Taherinejad, "An approximate carry disregard multiplier with improved mean relative error distance and probability of correctness," in *Euromicro Conference on Digital Systems Design 2022 (DSD2022)*, 2022, pp. 1–7.

[3] W. Wang, Y. Yang, X. Wang, W. Wang, and J. Li, "Development of convolutional neural network and its application in image classification: a survey," *Optical Engineering*, vol. 58, no. 4, p. 040901, 2019.

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

[5] N. TaheriNejad and S. Shakibhamedan, "Energy-aware adaptive approximate computing for deep learning applications," in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 328–328.

[6] S. E. Fatemieh, M. R. Reshadinezhad, and N. TaheriNejad, "Fast and compact serial imply-based approximate full adders applied in image processing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 13, no. 1, pp. 175–188, 2023.

[7] N. Amirafshar, A. S. B., H. S. Shahhoseini, and N. Taherinejad, "Carry disregard approximate multipliers," pp. 1–14, 2023.

[8] 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.

[9] R. Molina-Robles, A. Arnaud, M. Miguez, J. Gak, A. Chacón-Rodríguez, and R. García-Ramírez, "An energy consumption benchmark for a low-power risc-v core aimed at implantable medical devices," *IEEE Embedded Systems Letters*, 2022.

[10] H. E. Ghor, M. Chetto, and R. E. Osta, "Multiprocessor real-time scheduling for wireless sensors powered by renewable energy sources," in *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, 2018, pp. 1–6.

[11] M. Chetto, "Optimal scheduling for real-time jobs in energy harvesting computing systems," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 2, pp. 122–133, 2014.

[12] A. S. Baroughi, S. Huemer, H. S. Shahhoseini, and N. TaheriNejad, "AxE: An approximate-exact multi-processor system-on-chip platform," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 60–66.

[13] 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.

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

[15] 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.

[16] J. D. Lin, A. M. K. Cheng, and G. Gercek, "Partitioning real-time tasks with replications on multiprocessor embedded systems," *IEEE Embedded Systems Letters*, vol. 8, no. 4, pp. 89–92, 2016.

[17] J. Lin and A. M. Cheng, "Real-time task assignment with replication on multiprocessor platforms," in *2009 15th International Conference on Parallel and Distributed Systems*, 2009, pp. 399–406.

[18] 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.

[19] C. Wolf. Yosyshq/picorv32: Picorv32 - a size-optimized risc-v cpu. Last Accessed: June 22, 2023. [Online]. Available: https://github.com/YosysHQ/picorv32

[20] Faculty of information technology Bozetechova. Evoapproxlib — approximate circuits library — 8-bit unsigned multiplier. Last Accessed: June 22, 2023. [Online]. Available: https://ehw.fit.vutbr.cz/evoapproxlib/

[21] 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.