TaDA: Task Decoupling Architecture for the Battery-less Internet of Things

Weining Song
weining.song@angstrom.uu.se
Uppsala University
Sweden

Stefanos Kaxiras
stefanos.kaxiras@it.uu.se
Uppsala University
Sweden

Thiemo Voigt
thiemo.voigt@angstrom.uu.se
Uppsala University, RISE
Sweden

Yuan Yao
yuan.yao@it.uu.se
Uppsala University
Sweden

Luca Mottola
luca.mottola@polimi.it
Politecnico di Milano
Italy

ABSTRACT

We present TaDA, a system architecture enabling efficient execution of Internet of Things (IoT) applications across multiple computing units, powered by ambient energy harvesting. Low-power microcontroller units (MCUs) are increasingly specialized; for example, custom designs feature hardware acceleration of neural network inference, next to designs providing energy-efficient input/output. As application requirements are growingly diverse, we argue that no single MCU can efficiently fulfill them. TaDA allows programmers to assign the execution of different slices of the application logic to the most efficient MCU for the job. We achieve this by decoupling task executions in *time and space*, using a special-purpose *hardware interconnect* we design, while providing persistent storage to cross periods of energy unavailability. We compare our prototype performance against the single most efficient computing unit for a given workload. We show that our prototype saves up to 96.7% energy per application round. Given the same energy budget, this yields up to a 68.7x throughput improvement.

KEYWORDS

Task decoupling, Internet of Things (IoT), energy harvesting, intermittent computing

1 INTRODUCTION

To abate maintenance costs, ambient energy harvesting replaces traditional batteries to power Internet of Things (IoT) devices [8, 16, 22, 32]. However, energy from the environment is generally erratic, causing frequent and unanticipated energy failures. Increasingly diverse application requirements [16] and the vast array of available microcontroller units (MCUs) [4, 5, 7] further complicate matters.

Applications and MCUs. IoT applications blend diverse requirements, especially relative to energy consumption [48]. For example, input/output (I/O) operations through sensors and actuators enable interactions with the environment. The corresponding energy demands depend on peripherals' characteristics and the features of the computing unit controlling them. Data processing using signal processing [53] heavily depends on the computing unit's ability to handle floating-point operations. Embedded neural network inference [28] requires efficient multiply-accumulate computations (MACs). The energy performance of wireless transmissions depends on the coupling between the MCU and the RF transceivers.

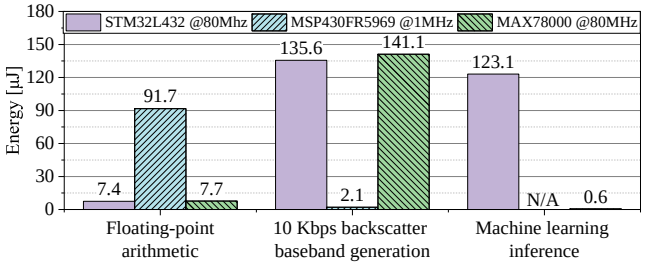


Figure 1: Energy consumption of different MCUs for given workloads. *No single MCU can fulfill the diverse requirements of IoT applications.*

No single available MCU fulfills these requirements alone. Fig. 1 illustrates key examples. When executing floating-point operations, a 32-bit higher-power MCU with a hardware floating-point unit (FPU) is ultimately more energy efficient compared to a lower-power MCU with FPU emulation [33]. The MSP430FR5969 MCU running at 1 MHz only consumes 0.4 mW compared to the Cortex M4 STM32L432 MCU running at 80 MHz, which absorbs 25 mW. To complete the same floating-point arithmetic workload, however, the MSP430FR5969 MCU requires 12 times the energy of the STM32L432 MCU, due to longer processing times.

The opposite observation applies when using the same two MCUs to achieve low-power communications with technologies such as backscattering [49], which is an asset with energy harvesting [11]. As ambient backscatter usually employs very low data rates [35] and computations for carrier modulation are extremely simple, and yet must occur at the same rate as data transmissions, a low-power MCU usually provides better energy performance. Fig. 1 indicates, for example, that generating the same baseband signal for 10 Kbps backscatter communications using the STM32L432 MCU running at 80 MHz consumes 65 times more energy than using the MSP430FR5969 MCU running at 1 MHz.

Special-purpose accelerators and custom instruction set architectures [20] achieve better energy efficiency than general-purpose MCUs for specific workloads. For example, the MAX78000 and STM32L432 MCUs are both based on a Cortex M4 core but the former features a specialized accelerator for neural network inference. To complete the same inference process, the STM32L432 MCU consumes 205 times more energy than the MAX78000 MCU, with both

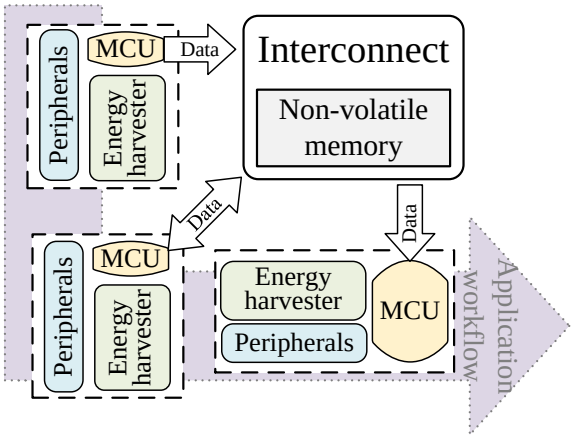


Figure 2: TaDA overview. *Developers provide the application code as a sequence of processing tasks running with transactional semantics. Tasks exchange data asynchronously through a hardware interconnect that also provides persistent storage.*

platforms running at the same clock. In Fig. 1, the energy consumption of the MAX78000 MCU in this case is so much lower compared to the STM32L432 MCU that the green bar is barely visible.

TaDA. Our goal is to enable that given functionality is executed on the best-performing MCU for that specific job. The best-performing MCU is the one that consumes the least energy for the specific workload, or for a specified slice of it, compared to the other available MCUs. For example, as shown in Fig. 1, the MAX78000 MCU is the best-performing MCU for machine learning inference.

To accomplish this, we design a system architecture that allows IoT applications to execute *across* a heterogeneous set of MCUs. We call it TaDA. Fig. 2 illustrates its fundamental components. Developers provide the application code as a sequence of processing tasks [17, 36, 38] with defined inputs and outputs and executing with *transactional semantics*, based on existing programming models [17, 36, 38]. With TaDA, the developer-provided tasks execute over separate memory spaces of the different MCUs and only exchange data *asynchronously* through a special-purpose *hardware interconnect* providing a message-passing interface. Task execution is thus decoupled *in space*.

Battery-less IoT applications are also prone to energy failures. Persisting intermediate results is necessary to cross periods of energy unavailability, leading to an *intermittent computing* pattern [11]. The use of non-volatile memories (NVMs) to this end is challenging; as of now, only one family of commercially-available MCUs provides dedicated support [5]. We equip TaDA's hardware interconnect with NVM to persist messages flowing between MCUs, providing a means to intermittent computing also for platforms with no built-in support for handling energy failures. MCUs pushing (pulling) data to (from) the interconnect, therefore, may not be active at the same time and be powered by separate energy harvesters. Task execution is therefore also decoupled *in time*.

We prototype TaDA's interconnect using an MSP430FR5969 MCU with on-chip NVM support and a small software driver, amounting to only around 1000 lines of C code. Using real hardware, we concretely implement three staple IoT applications, namely human activity recognition (HAR), plant health monitoring (PHM), and agriculture environment measurement (AEM). Each application uses different sensing and communication technology, exposes different processing and data requirements, and therefore employs a different combination of MCUs around the TaDA interconnect.

As the interconnect itself bears an energy overhead, we investigate whether that potentially cancels the gains obtained by running each task on the best-performing MCU. We run the three applications on top of the TaDA prototype to compare their energy and throughput performance with their execution on the best-performing *single* MCU. This is the one that consumes the least energy for each of the three applications compared to the other available MCUs, when the application runs entirely on that MCU. We find that TaDA saves up to 96.7% energy compared to the best-performing single MCU in a single application round. Using a total of 32 hours of 20 different real-world power traces, we demonstrate that TaDA enables up to a 68.7x throughput improvement.

In summary, this paper presents three key contributions:

(1) we design a *system architecture* providing space- and time-decoupled execution across different MCUs of IoT applications powered by energy harvesting;
(2) we create a *prototype* using off-the-shelf hardware and a thin software layer we develop, and use this to implement three staple IoT applications;
(3) using real hardware, we show up to *96.7% energy gains* compared to the best-performing single MCU for the same workload, corresponding to a *68.7x throughput improvement* measured using real-world power traces.

The paper unfolds as follows. Sec. 2 provides background information and contrasts our work with related efforts. Sec. 3 describes the design and implementation of TaDA, whereas Sec. 4 describes the three application prototypes. Sec. 5 illustrates the evaluation settings and results. We end the paper in Sec. 6 with a discussion on limitations and in Sec. 7 with brief concluding remarks.

2 BACKGROUND AND RELATED WORK

Our work is loosely inspired by the concept of decoupled access-execute (DAE), originally proposed by Smith [46] as a means to improve execution performance by decoupling memory access and execution in general-purpose computing [34]. Rather than focusing on specific operations of the same application, TaDA decouples different slices of the application code in space and time using an NVM-equipped hardware interconnect, allowing workload-specific MCUs asynchronously exchange data. We provide additional background and briefly survey related works next.

Intermittent computing. Battery-less IoT devices use ambient energy sources, such as solar, thermal, and radio frequency (RF), as their only power source [16]. However, due to erratic energy patterns, executions become *intermittent*: intervals of active operation are interleaved by periods of recharging energy buffers [11].

Battery-less IoT devices typically feature 16- or 32-bit MCUs with a few kilobytes of memory. Applications run on bare hardware with no operating system. Energy failures normally cause a device to lose system state. To ensure forward progress across energy failures, systems employ *persistent state* on NVM, which is

restored when energy is back, so executions resume close to the point of energy failure rather than performing a complete reboot. The energy required for NVM operations, however, may reach up to 350% of the cost of the application processing [50]. Some solutions employ a form of checkpointing to save the intermediate program state to NVM [10, 12, 13, 15, 39, 45, 50]. This approach replicates the application state on NVM at specific points in the program, and restores the state once the system has sufficient energy to continue. Existing techniques differ based on the amount of operation accomplished at compile- versus run-time, and by the logic used to place checkpoints in the code.

Other approaches offer a *task* abstraction to define and manage persistent state [17, 36, 38]. A task is a computational unit with defined inputs and outputs, which runs with transactional semantics and commits outputs on NVM. Tasks allow one to precisely map the energy consumption to the available hardware, for example, when using multiple capacitors [19]. Using existing techniques [18, 19], the capacitor array may be configured, in number and size of individual capacitors, to ensure eventual completion of the workload.

MCUs. Designs exist with built-in NVM support, for example, the TI MSP430FRxxxx series [5]. Using these MCUs, the software support only requires to checkpoint a few processor registers to NVM [40]. Because of this feature, these MCUs are often the basis for intermittent computing platforms [30] and deployments [8].

Ma et al. also present a design that uses NVM throughout the entire memory hierarchy, down to flip-flops [37]. Checkpoints are small but frequent, allowing the system to resume computation quickly after energy failures. Programmers no longer need to manage state persistence operations explicitly in the code but hardware complexity and energy overhead increase.

In TaDA, we seek to reap the greatest benefits from existing MCUs regardless of their built-in support to intermittent computing. We let developers use the best-performing MCU for the job depending on the requirements of specific slices of the application logic. TaDA also allows general-purpose MCUs to be used to compute intermittently by providing persistent storage.

Multiple MCUs. Modern System-on-Chip (SoC) devices package multiple MCUs in the same integrated circuit. For instance, the TI C13xx/CC26xx series [1] integrate two low-power MCUs besides the main one. These MCUs are constrained to specific tasks, such as sensing and wireless communication, and cannot be programmed directly. The lack of shared memory requires the main MCU to be active for any inter-MCU communication.

Other MCUs integrate two general-purpose computing cores, such as the ARM Cortex-M4F and the ARM Cortex-M0+ in the NXP's K32 L Series MCUs [2]. A specific messaging unit (MU) module allows the two cores within the MCU to exchange data. Here again, this design is limited to these exact two computing cores, which work in the same power and clock domains by construction. They cannot be decoupled in time, as required by unpredictable ambient energy patterns that may create the conditions for only one of the two cores to be active at a time.

Modular architectures exist, including the 4-layer modular architecture [44], the stackable architecture [43], the MIT Media Lab modular platform [14], and Epic [23]. In these designs, the components communicate through a shared bus or standard communication protocols. They can accommodate more than two MCUs but again lack shared memory in between for storing data. The MCUs in these designs must be powered on at the same time to exchange data with each other. Time decoupling, as above, is unfeasible.

Closer to our work is Bolt [47], a dual-MCU platform that supports time-sensitive communication between MCUs. Its design is originally motivated by the necessity to handle synchronous communications [24] without halting the regular application processing. To this end, BOLT decouples an application-specific MCU from a communication-specific MCU, yet while guaranteeing that time constraints necessary for synchronous communications are not violated. These guarantees are formally verified properties using Uppaal models, which are created for the two specific MCUs. Because of the time-sensitive setting, the two MCUs know exactly when to push or pull data.

In contrast, we focus on decoupling intermittent computing tasks that have no predictable time dynamics executing on independently-powered MCUs. Rather than a specific two-MCU setting, moreover, our design is general in that it accommodates an arbitrary number of different MCUs, which are not necessarily fixed and may vary depending on application requirements.

List 1 TaDA asynchronous message-passing API

```
1: void push (uint8_t* message, uint8_t size);
2: void pull (uint8_t* message, uint8_t size, uint8_t id);
3: uint8_t status (uint8_t id);
```

3 TADA

We describe the TaDA programming model, our prototype implementation, and the system configuration.

3.1 Programming Model

Developers provide applications encoded as an acyclic graph with tasks represented as nodes in the graph. Tasks have defined inputs and outputs connecting them together and execute with transactional semantics. This application encoding is the same as existing task-based programming abstractions for intermittent computing [17, 36, 38, 52]. Tools exist to guide developers in the process of decomposing existing application code into tasks [18].

List 1 shows the asynchronous message-passing API that the tasks deployed on individual MCUs use to exchange data with other MCUs. A small software driver implements this API for the specific MCU. The API allows the single MCU to push (pull) data to (from) the interconnect and to probe its status, enabling the execution of the tasks deployed on different MCUs on separate memory spaces. This effectively decouples task executions *in space*.

Tasks deployed on different MCUs and pushing (pulling) data to (from) the interconnect may not execute at the same time or with any specific synchronization. The interconnect buffers messages over persistent storage in *pairwise FIFO order*. This way, regardless of energy patterns, task executions are decoupled *in time* as well.

An arbitrary number of MCUs may attach to the interconnect. The `id` parameter in List 1 is used to specify what MCU to pull from.

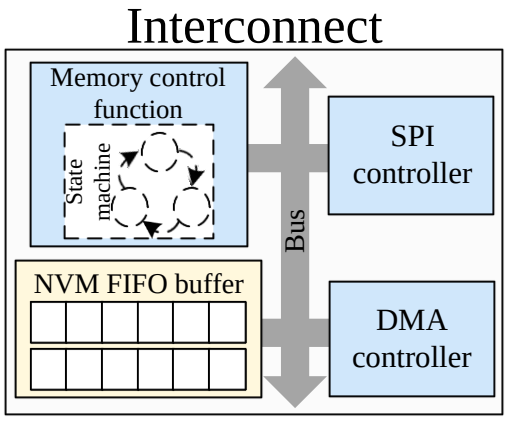


Figure 3: Interconnect implementation. *The MSP430FR5969 MCU provides persistent storage on FRAM. We use the SPI to handle data exchanges with MCUs.*

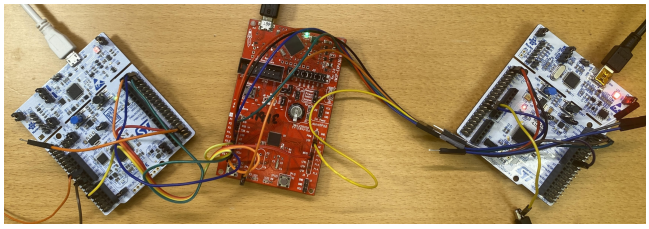


Figure 4: TaDA hardware prototype. *The interconnect prototype in the middle connects two different MCUs.*

Moreover, whenever a TaDA-attached MCU starts the execution, the interconnect is not necessarily ready to handle the data transfers. For example, it might not have storage space to accommodate the data. Hence, a *pre-execution* stage is necessary to ensure the interconnect can handle the requests of the attached MCU before task execution, as further detailed in Sec. 3.3. The **`status()`** operation of List 1 returns information on the interconnect status.

3.2 Prototype

We prototype TaDA around a custom interconnect using off-the-shelf hardware. This is not the only possible implementation; for example, one may design a custom circuit providing the same semantics. Our prototype provides a quick path to a working implementation. It also demonstrates limited overhead and enables significant overall performance gains, as we report in Sec. 5.

We use an MSP430FR5969 MCU, a low-power MCU with built-in FRAM, for the interconnect prototype. It provides efficient operation especially when handling peripheral interactions. The built-in 64KB FRAM facilitates implementing persistent storage semantics with low energy overhead. The MCU runs a custom memory controller we implement in C. The controller manages a variable number of FIFO queues stored in FRAM. We use one FIFO queue for every pair of upstream-downstream MCUs that must exchange data. This is configured at compile-time.

The software driver that implements the API of List 1 uses the built-in serial peripheral interface (SPI) and the direct memory access (DMA) controller of the MSP430FR5969 MCU to enable efficient data exchanges between the MCUs and the interconnect. SPI can achieve higher data rates compared to alternatives like I2C and UART. The DMA controller enables faster and more efficient data exchanges by moving data directly between MCUs and the interconnect without MCU intervention. We handle the control signals indicating the status of the FIFO queues using the general-purpose input/output (GPIO) pins.

Each MCU is equipped with a dedicated capacitor. However, the interconnect does not have its own capacitor; it is directly powered by the capacitor powering the MCU that is currently using it. This design ensures that the interconnect is active at the same time as the MCU attempting a data exchange with it.

3.3 System Configuration

We size the capacitors based on the peak energy demand of the tasks assigned to that MCU and on its active power, using the same principles as in existing multi-capacitor architectures [19]. The capacitors size is specifically determined to ensure that the attached MCU has sufficient energy to sustain the worst-case processing demands of the most energy-demand task, plus the required data exchanges with the interconnect.

The capacitors may be charged in different ways. One may attach them in parallel with a single energy harvester, which entails the capacitors are charged proportionally to their size. Alternatively, one may design a simple circuit that distributes incoming energy equally among the capacitors, or attach each capacitor to a different energy harvester. We evaluate the effect of the former two charging strategies in Sec. 5. We disregard the latter in that it increases the total energy budget available to the system, and hence is not comparable with any other configurations.

The pre-execution stage also requires energy. Two possible configurations are available here. We call one Check-Only. With this, we factor into the estimate of the capacitor size described earlier a minimal amount of additional energy. The MCU uses the additional energy to execute the pre-execution stage. If the interconnect cannot handle the requests, the MCU shuts down and waits for the next time it can activate.

A different design option, instead, entails reserving some additional energy to perform repeated checks during the pre-execution stage. This essentially requires selecting a capacitor storing energy beyond what is strictly needed by task execution and data exchanges with the interconnect. For example, say a task requires 10 μJ to complete. One may add a 20% fraction of energy to perform the pre-execution, ultimately using a capacitor worth 12 μJ. In this case, if the interconnect cannot handle the request, the MCU enters sleep mode and waits for an interrupt from the interconnect through GPIO, which wakes it up and triggers another attempt. The process repeats for a number of times limited by the additional energy budget, that is, 2 μJ in this example. Using more energy would put at risk the completion of the task, should the interconnect be eventually ready to handle the request.

4 APPLICATION PROTOTYPES

We develop three staple IoT applications to study the performance of a TaDA-based full-fledged implementation, compared to a single-MCU counterpart. We begin by manually partitioning the application code in tasks, isolating key functionality such as sensing, processing, and wireless communications in separate tasks, similar to how an application developer using TaDA operates.

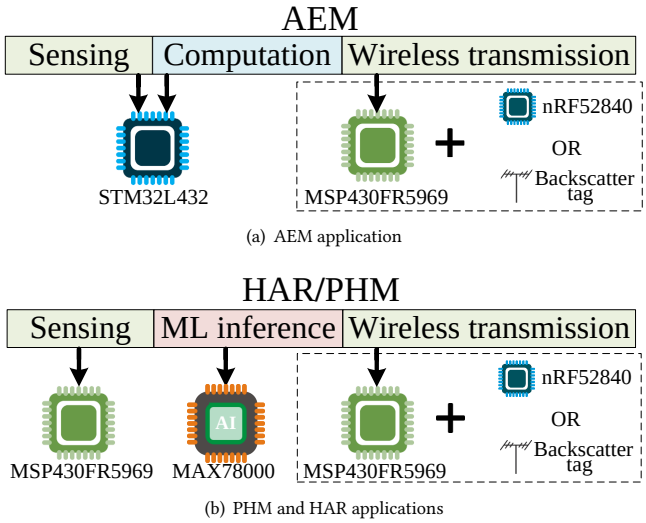
To determine the deployment configuration of TaDA for each application, including the number of different MCUs to employ, what MCU runs what tasks, and the interconnect configuration, we profile the application processing using existing tools [9, 25, 30]. Based on this information, we compute the energy that different functionality require depending on the target MCU and attached peripherals, if any, among the three MCUs we discussed in Sec. 1 and by considering the application-specific sensors. These MCUs are representative of different trade-offs, in that they include a low-power MCU, a high-power MCU with hardware FPU, and a special-purpose hardware accelerator. We return to the choice of the three specific MCUs we consider in this work in Sec.6. The measured energy consumption of each task within an application on each MCU is compared to identify the MCU that executes that specific task with the least energy consumption. This MCU is considered the best-performing MCU for that specific task.

We also explore two wireless technologies, namely, BLE [31] and backscatter communications [35]. We use the nRF52840 chip as the BLE transceiver, which works in advertising mode to send data. The on-board Cortex M core does not run any slice of the application logic and only runs the BLE stack. We use the MSP430FR5969 MCU for backscatter baseband signal generation and test the two different data rates. The encoding functionality we mention hereafter only applies to backscatter. We attach a simple backscatter tag to the MSP430FR5969 MCU to enable data transmissions.

Agricultural environment measurement (AEM). AEM senses the environmental temperature and humidity for monitoring of greenhouses [21]. Data is locally processed first and then relayed to a central collection point for processing, using wireless communication. This is representative of a large class of environment monitoring IoT applications [42], including many that employ energy harvesting [16].

Fig. 5(a) shows the tasks used to encode the application processing. Based on profiling information, we only use two MCUs: we deploy a task including sensing and processing onto an STM32L432 MCU, whereas packet preparation, encoding and transmissions are deployed onto an MSP430FR5969 MCU. This configuration is natural in that the MSP430FR5969 MCU provides energy-efficient I/O operations. In contrast, floating-point arithmetic is required to process temperature and humidity readings, which is where the STM32L432 MCU excels compared to the alternatives. We dimension the FIFO buffer at the interconnect to accommodate 700 data items of 32 bits each, which leaves as much room as possible to handle data bursts, as we further discuss in Sec. 5.

Plant health monitoring (PHM). PHM is an IoT application that classifies plant diseases [6, 26]. An embedded camera takes pictures of the leaves; a machine-learning (ML) model is used to infer the plant status. The outcome is eventually sent to a central collection point using wireless. The processing is a paradigmatic example of handling large data on IoT devices, in that images impose quite a significant memory overhead, and of running ML functionality [29].



(a) AEM application

(b) PHM and HAR applications

Figure 5: Tasks encoding for three applications. *We use an MSP430FR5969 MCU to handle I/O with peripherals and RF transceivers, a MAX78000 MCU to run inference processes, and an STM32L432 MCU to perform floating-point calculations.*

We split the application processing in three tasks, shown in Fig. 5(b). Profiling information leads us to employ all three MCUs. We use an MSP430FR5969 MCU to drive the camera operation, that is, the sensing task. Packet preparation and execution of the inference step are deployed onto a MAX78000 MCU. As for the AEM application, encoding and transmission functionality are deployed to a (different) MSP430FR5969 MCU. The rationale is that an MSP430FR5969 MCU is the most efficient, among the alternatives we consider, to handle I/O operations with sensors and transceivers, whereas the MAX78000 MCU provides acceleration for inference.

The FIFO buffer between the MSP430FR5969 MCU driving the camera and the MAX78000 MCU is set to accommodate up to 15 data items, each of 64x64x8 bits, which is the maximum our hardware prototype can support due to memory constraints. The other FIFO buffer between the MAX78000 MCU and the MSP430FR5969 MCU is set to provide space for 15 data items of 32 bits each.

Human activity recognition (HAR). HAR is an IoT application that categorizes human actions based on high-frequency accelerometer data [6, 51], using an ML classification model. As in the other cases, wireless communications are used to relay the outcome to a central collection point. The processing at hand is another example of running ML functionality at the edge [29], yet this time the individual samples are small but acquired at higher frequency.

Fig. 5(b) shows the three tasks we use to encode the application logic. An MSP430FR5969 MCU drives the accelerometer and buffers incoming data. As in the PHM application, an MAX78000 MCU runs the packet preparation and inference step, whereas a (different) MSP430FR5969 MCU handles encoding and transmissions. The rationale is similar as above. This configuration, however, also

shows how TaDA allows the same accelerator to handle different types of data independent of how they are generated. In both the HAR and PHM applications, the MSP430FR5969 MCU upstream prepares the data for the MAX78000 MCU to proceed. It can do that much more efficiently than the MAX78000 MCU itself, which mainly accelerates neural network inference. We dimension each of the two FIFO buffers at the interconnect to accommodate 700 data items of 32 bits each.

5 EVALUATION

Our evaluation is three-pronged. Sec. 5.1 investigates the energy performance of the three prototype applications in Sec. 4, based on real hardware measurements. Sec. 5.2 studies how the energy savings enabled by TaDA in a single application round unlock many-fold throughput improvements, using real-world power traces that ensure reproducibility. We report on the execution of several micro-benchmarks in Sec. 5.3, which are instrumental to understand the impact of parameter settings and the architecture's limitations. Our results indicate that

(1) a TaDA-based implementation of the applications we consider enables up to a 96.7% improvement compared to the best-performing single MCU;
(2) the prototype of TaDA interconnect bears limited energy overhead, which amounts to 2.6% of the total energy consumption in the worst case;
(3) depending on energy patterns, a TaDA-based implementation can complete 68.7x more application rounds than the single MCU configuration.

5.1 Energy

We describe the setup of the energy measurements and the results we obtain. The energy improvements are the basis to enable the throughput improvements we illustrate next.

Setup. We use the hardware/software prototypes in Sec. 4 to run the application code and measure its *energy consumption* with real hardware. To do so, we attach the hardware prototypes to a digital power supply [3], which provides a fixed voltage to the hardware prototypes and records their power consumption over time. We can calculate the energy consumption of the hardware prototypes based on the power consumption traces we obtain.

We compare the energy consumption of a TaDA-based implementation of three applications with the best-performing *single* MCU implementation. To identify the best-performing single MCU for each application, we experimentally measure its energy consumption when running on each of the three MCUs of Sec. 1. The processing task in the AEM application is floating-point arithmetic, which is best served by an STM32L432 MCU as shown in Fig. 1. However, because wireless communication involves long MCU idle time, we will obtain the best energy performance for the AEM application with a single MSP430FR5969 MCU because of its low power consumption. The MSP430FR5969 MCU is not practical for applications including machine learning functionality, that is, PHM and HAR, due to hardware limitations and the lack of toolchain support for machine learning inference. In contrast, the MAX78000 MCU achieves the best energy performance for machine learning inference compared to the STM32L432 MCU. Since both the MAX78000 MCU and STM32L432 MCU exhibit similar power consumption, their energy consumption for wireless communication is also similar. As a result, the MAX78000 MCU is the best-performing single MCU whenever machine learning inference is needed, that is, in the PHR and HAR applications.

For communications, we explore both the use of BLE using the nRF52840 transceiver and of backscatter communication at 10Kbps or 1Kbps using an MSP430FR5969 MCU for encoding the baseband signal, as discussed in Sec. 4.

Results. Fig. 6 shows the energy figures we obtain. The TaDA-based implementations consistently outperform the corresponding single-MCU configurations.

As shown in Fig. 6(b), we achieve the best performance improvement when using TaDA for the HAR application with 1 Kbps backscatter communication, yielding a 96.7% energy improvement compared to the single MAX78000 MCU. The latter is the best-performing single MCU for the specific workload and yet consumes much energy when generating the baseband signal for backscatter communication, because of long processing times and high power consumption. Much of the gain here originates from the ability of offloading signal generation onto an MSP430FR5969 MCU. Given the processing times are fixed, this abates the corresponding energy consumption due to lower power consumption.

The lowest, still significant energy gain is obtained with the PHM application using BLE communication. Compared to the single MAX78000 MCU, which is the best-performing single MCU for the specific workload, the TaDA-based implementation cuts almost a third of the energy consumption. In this case, the gain is due to the ability of employing an MSP430FR5969 MCU to perform the sensing task, especially including driving the embedded camera.

Fig. 6 also depicts the contribution of each MCU attached to the TaDA interconnect for every configuration, along with the energy consumption of the interconnect itself. Crucially, the interconnect's energy consumption is immaterial across all configurations we test, to the point of being barely visible in the charts. The worst-case energy consumption due to the interconnect is 2.6% of the total.

For the AEM application, shown in Fig. 6(a), the major contribution to the total energy consumption are the STM32L432 MCU handling floating-point calculations for signal processing and either the MSP430FR5969 MCU or the nRF52840 transceiver handling backscatter and BLE communications, respectively. In the HAR application, shown in Fig. 6(b), the contribution to the total energy consumption of the three MCUs we used is roughly even. The only noticeable difference is the impact of the MSP430FR5969 MCU used to drive backscatter communications, which consumes much more in the 1 Kbps configuration because of longer packet transmission times, as expected. The sensing task using the MSP430FR5969 MCU vastly dominates energy consumption in the PHM application, shown in Fig. 6(c), whereas the contribution of the MCU handling communications is negligible.

Worth observing is also that each MCU attached to the interconnect incurs an additional energy overhead in TaDA that would not exist if the system ran in a single MCU configuration. This is the energy cost for pushing (pulling) data to (from) the interconnect through the SPI, that is, to use the API in List 1. For example, this overhead using the MAX78000 MCU is 29% and 9.8% of the total

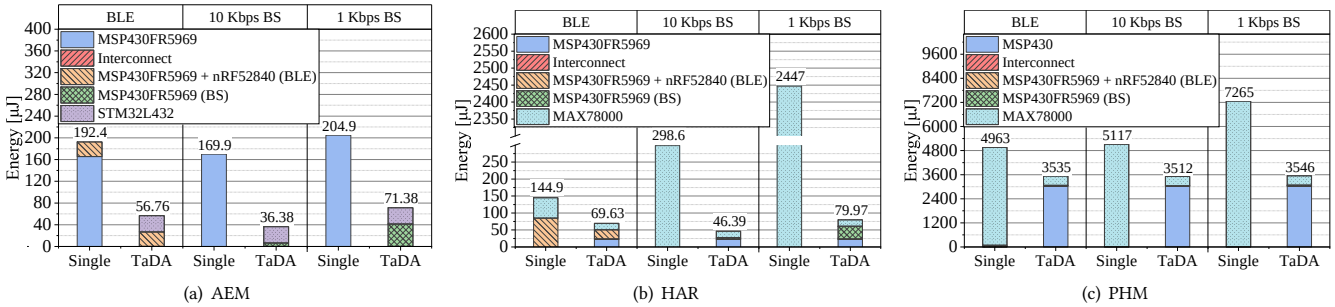


Figure 6: Energy consumption of AEM, HAR, and PHM running on TaDA, compared with the best-performing single MCU, using different communication technologies. *A TaDA-based implementation saves up to 96.7% of the energy consumption compared to a single MCU. The interconnect prototype bears a maximum 2.6% energy overhead.*

energy consumption when running the PHM and HAR applications, respectively. It is 10% of the total energy consumption using the STM32L432 in the AEM application. Notwithstanding this overhead, the TaDA configuration still consumes farless energy compared to the best-performing single MCU.

5.2 Throughput

Given the same energy budget, understanding how the energy gains due to TaDA allow systems to process more data requires accounting for the environment dynamics.

Setup. We measure the *number of completed application rounds* as a metric to evaluate system throughput over a fixed time period. We count a completed application round when a data item fully traverses the task pipeline, with each task being executed on the respective MCU. As the power supply from the environment fluctuates over time, the throughput reflects how efficiently TaDA utilizes the incoming energy as it becomes available. The energy from the environment in this amount of time is fixed and the same for both TaDA-based implementations and single-MCU configurations, both in absolute values and also over time.

We use 20 different real-world power traces [27]. These are collected from five diverse scenarios, including an indoor office, outdoor stairs, a car moving, people jogging, and a washing machine operating. Several solar panels are deployed in the indoor offices and outdoor stairs. Several piezoelectric harvesters are deployed on the car and on the washer machine. Multiple solar panels and piezoelectric harvesters are deployed on two different people jogging. The different features of the power traces we employ, also discussed next, provide support to the generality of our results.

Using these power traces, we synthetically execute the three applications in Sec. 4 using either the TaDA-based implementation or a single MCU. We carefully model the capacitor charging process and the execution of the application logic on the different MCUs and the interconnect. We obtain the energy consumption profile from the real hardware executions we discussed in Sec. 5.1, similar to existing work in intermittent computing [9, 11, 41].

We determine the size of the capacitors for the TaDA-based implementation as described in Sec. 3. The capacitor attached to the single-MCU configuration suffices to complete a single application round. We split energy from the power traces to charge the capacitors proportionally to their size. Each platform is using the Check-Only strategy to check the status of the interconnect as described in Sec. 3.

Results. Fig. 7, Fig. 8, and Fig. 9 show the number of complete application rounds for the TaDA-based implementation and the single-MCU configuration, given the same energy budget.

We achieve the maximum improvement when running the HAR application, shown in Fig. 8, with TaDA using 1 Kbps backscatter communication. This is expected, in that it corresponds to the highest energy gain as per the discussion in Sec. 5.1. The TaDA-based implementation can complete 68.7x more application rounds than the MAX78000 MCU alone, which is again the best-performing one among the possible single-MCU options. If the baseline were any other single MCU, the improvement would be even higher. Even with the minimum energy gain corresponding to the PHM application using BLE communication, shown in Fig. 9, the TaDA-based implementation doubles the number of application rounds completed with the same energy budget, compared to the MAX78000 MCU.

Note how the number of completed application rounds varies across different scenarios, depending on the power trace. For instance, node 1 in the jogging scenario exhibits the highest instant and average input power. This yields the highest number of completed application rounds overall for both the TaDA-based implementation and the single MCU configuration. When using the piezoelectric harvester on the car, the input power is fairly low both instantly and on average. The number of completed application rounds is thus reduced for both the TaDA-based implementation and the single-MCU configuration.

We investigate how input power impacts performance next, running micro-benchmarks that are instrumental to study the impact of parameter settings and energy patterns.

5.3 Micro-benchmarks

The functioning of a TaDA-based implementation depends on a few key system parameters and on external dynamics. We investigate both here. The setup is the same as in Sec. 5.2.

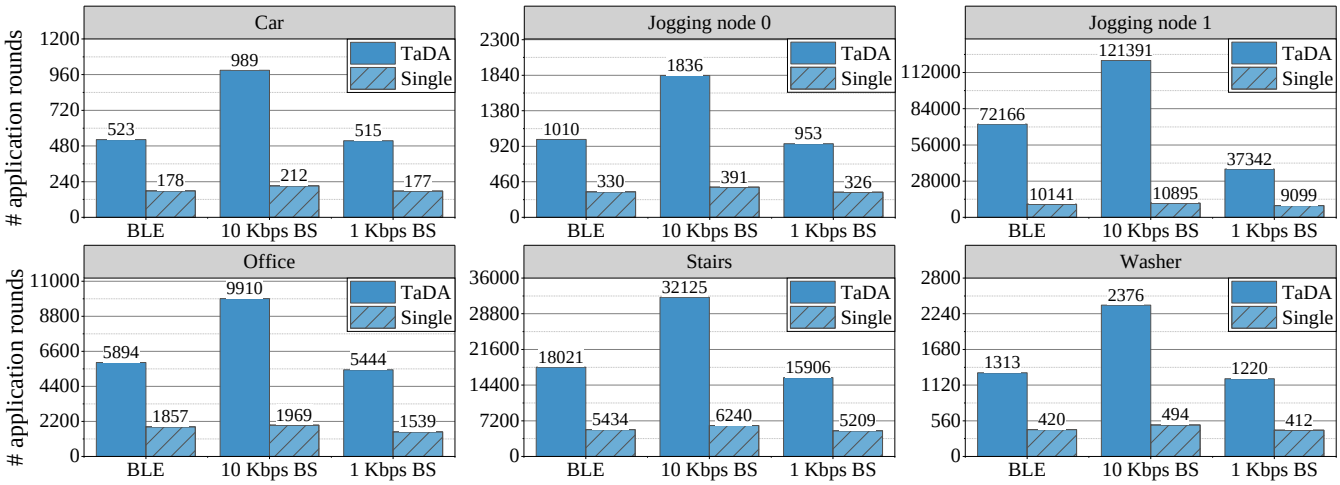


Figure 7: Completed application rounds for AEM application using TaDA, compared with the best-performing single MCU configuration, using different communication technologies. *Among all power traces, TaDA achieves better throughput improvement on Jogging node 1, indicating that TaDA can effectively utilize energy when the input power is high.*

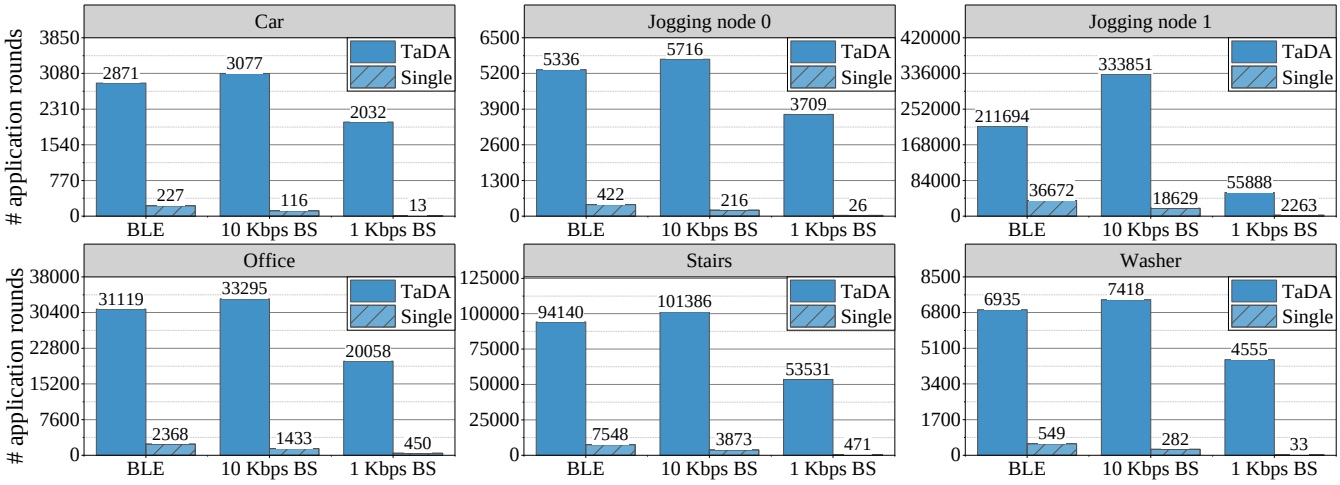


Figure 8: Completed application rounds for HAR application using TaDA, compared with the best-performing single MCU configuration, using different communication technologies. *In contrast to the AEM application, TaDA achieves the minimum throughput improvement on Jogging node 1 because the interconnect saturates in this scenario. We discuss these dynamics in Sec. 5.3.*

Pre-execution energy settings. We study how different energy configurations to execute the pre-execution stage, described in Sec. 3, may impact performance.

Fig. 10(a), Fig. 10(b), and Fig. 10(c) illustrate the number of completed application rounds across all power traces, using different settings of the checking strategy. The plot provides evidence that, in the applications we test and given the specific combination of hardware and energy patterns, adopting a Check-Only strategy achieves the best performance. This is especially visible when using 10 Kbps backscatter.

Inspection of the execution logs reveals that it happens very rarely that an MCU first queries the interconnect without finding data; then a subsequent check at a short time interval finds data instead. This means that tuning the pre-execution state to invest a variable percentage of energy for consecutive checks, while keeping an MCU in sleep mode, is almost never efficient and represents a waste of energy. Generally, however, the performance difference between the different settings is less pronounced with a higher number of completed application rounds, in that the relative impact of the energy waste becomes marginal.

Charging strategy. Based on the same power traces as in Sec. 5.2, we investigate how two of the possible charging strategies illustrated in Sec. 3 influence the performance. We disregard the configuration using multiple harvesters, in that this configuration would increase the total energy budget available to the system. The results would not, therefore, be comparable with other configurations.

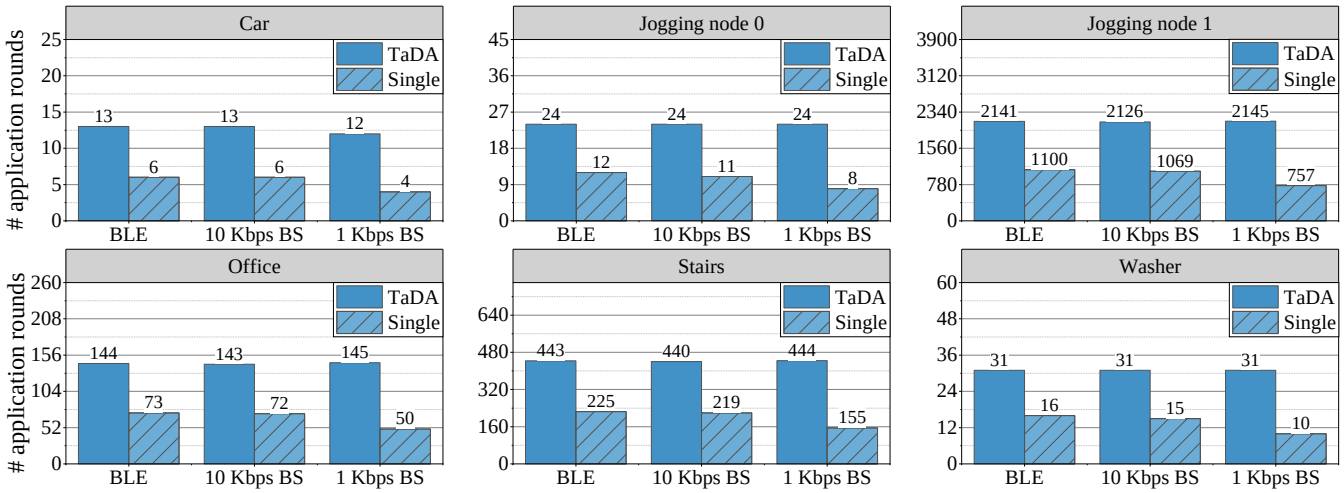
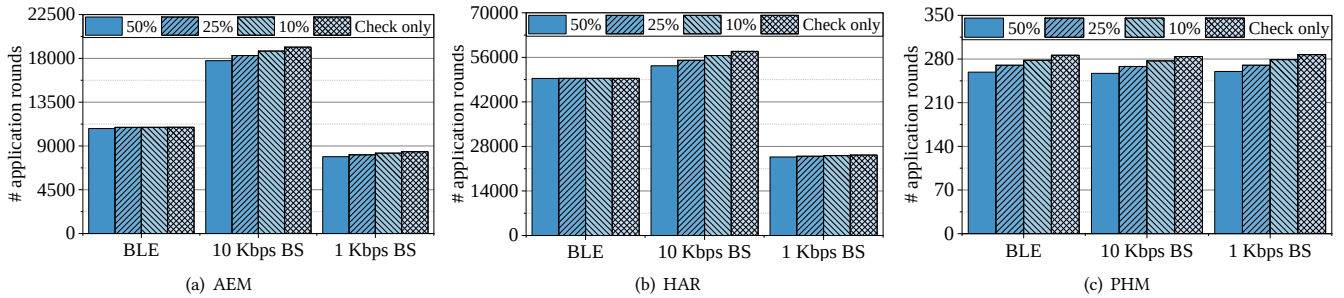


Figure 9: Completed application rounds for PHM application using TaDA, compared with the best-performing single MCU configuration, using different communication technologies. *Even with minimum energy gain in the case of BLE communication, TaDA still doubles the number of completed application rounds compared to the MAX78000 MCU alone.*



(a) AEM (b) HAR (c) PHM

Figure 10: Application rounds for TaDA-based implementations of AEM, HAR, and PHM with different energy configurations for the pre-execution stage. *Given the specific combination of hardware and energy patterns, the Check-Only strategy achieves the best performance.*

Fig. 11, Fig. 12, and Fig. 13 show the number of completed application rounds for the TaDA-based implementation by charging capacitors equally or proportionally to their size. The results indicate that the latter yields the best overall performance. Using this setting, it is rare to encounter situations where an MCU with the capacitor fully charged waits before executing, unless the interconnect is full or empty depending on whether it needs to pull or push. Energy is more effectively used this way.

Differently, charging capacitors through a dedicated circuit that equally distributes the incoming energy generates two undesired effects. As the capacitor size is proportional to the energy required to complete a workload, MCUs that require more energy are slowed down, in that they need to wait longer before reaching the activation threshold. MCUs that require little energy, on the other hand, are given more energy than what they really need. The interconnect consequently does more work and the average sojourn time of data at the interconnect increases, in that different MCUs are likely to be active at very different times.

Limiting factors. As the interconnect mediates the data exchanges across multiple MCUs in a TaDA-based implementation, it may potentially represent a bottleneck. We investigate the environment and system factors that may potentially limit the performance this way. We find that these include short-term average input power and fluctuations thereof, throughput of the communication technology employed, and size of the message buffer in the interconnect.

Deployments of battery-less IoT systems span a variety of energy sources [16], which may strikingly differ in power dynamics. Among the power traces we use, the most powerful one is the trace in the jogging scenario, which features both highest instant power (21 mW) and the highest average power (3.5 mW). The weakest power trace is the one from the piezoelectric harvester deployed on the washer machine, which shows both lowest instant power (1.3 mW) and lowest average power (0.08 mW). We note that the weaker is the power trace, the more the system throughput is mainly determined by that and not by other factors, which we

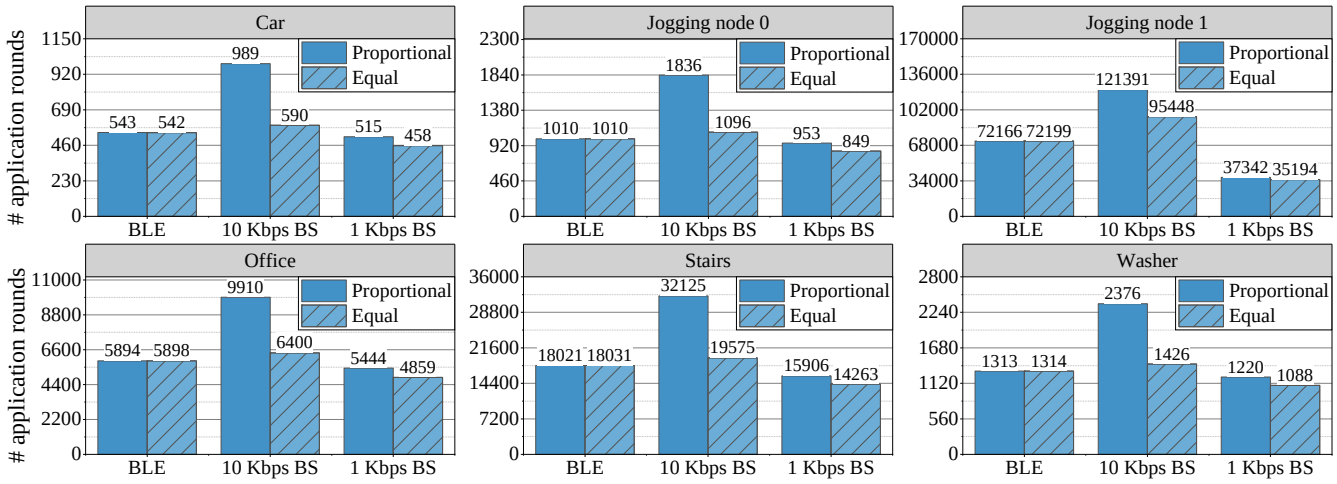


Figure 11: Application rounds for the TaDA-based AEM implementation, depending on charging strategies. *When using BLE communication technology, the two different charging strategies achieve similar throughput performance because the size of the two capacitors used in the TaDA-based implementation happens to be similar.*

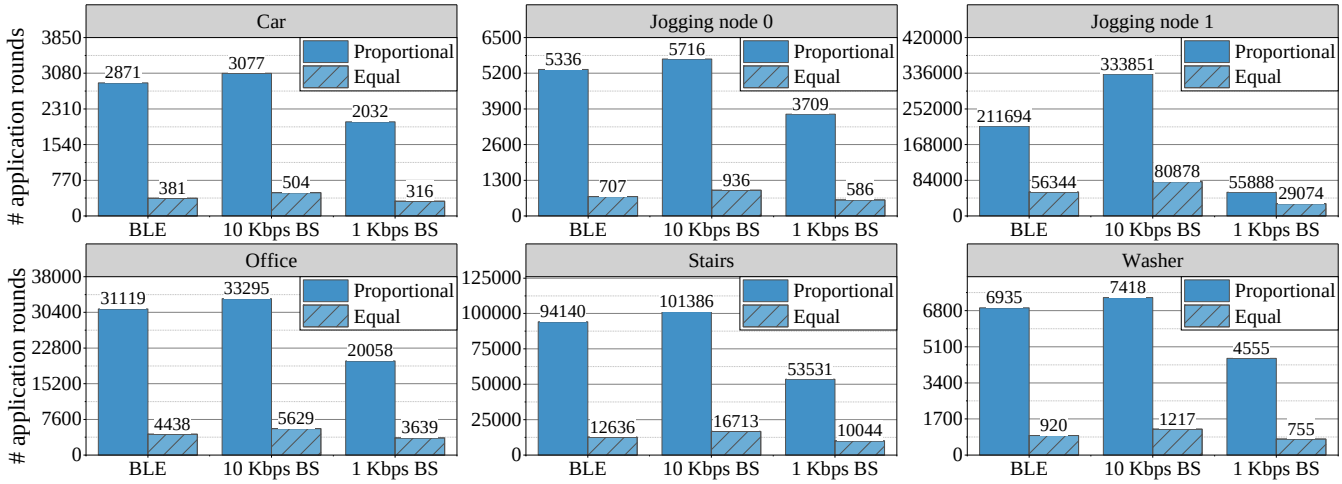


Figure 12: Application rounds for the TaDA-based HAR implementation with different charging strategies. *Charging capacitors proportionally to their size provides energy precisely where it is needed; in this case, to handle wireless communications*

discuss next. In a sense, the energy content of the environment is the first potential limiting factor.

Provided the ambient provides enough energy, the throughput offered by the communication technology is likely to represent the next limiting factor. The MCU in charge of driving transmissions is usually the last in the application workflow. When using backscatter communications with low data rates, for example, the time it takes for the communication technology to send data off might exceed capacitor recharge times. If so, data generated by MCUs upstream cannot be consumed in time and "piles up" at the interconnect.

We show evidence of this in Fig. 14, showing an excerpt of an execution of the AEM application using the jogging power trace with 1 Kbps backscatter communication. As the input power is high, yet backscatter communication at 1 Kbps is slow, the number of data items stored in the interconnect grows over time until it eventually reaches the limit, which is a system parameter as discussed in Sec. 3. In this situation, the execution stalls in that, even if the MCUs upstream have sufficient energy to produce more data, there is no space in the interconnect to store it. This situation occurs because the last MCU in the application workflow is too slow pulling data from the interconnect. When upstream platforms reach the activation threshold, they find no space in the interconnect to store more data and give up on executing.

Situations where the interconnect saturates may also happen transiently; for example, because of a sudden but temporary burst in input power. Independently of the communication technology, the number of items the interconnect can accommodate may become a limiting factor. For example, Fig. 15 shows a slice of the

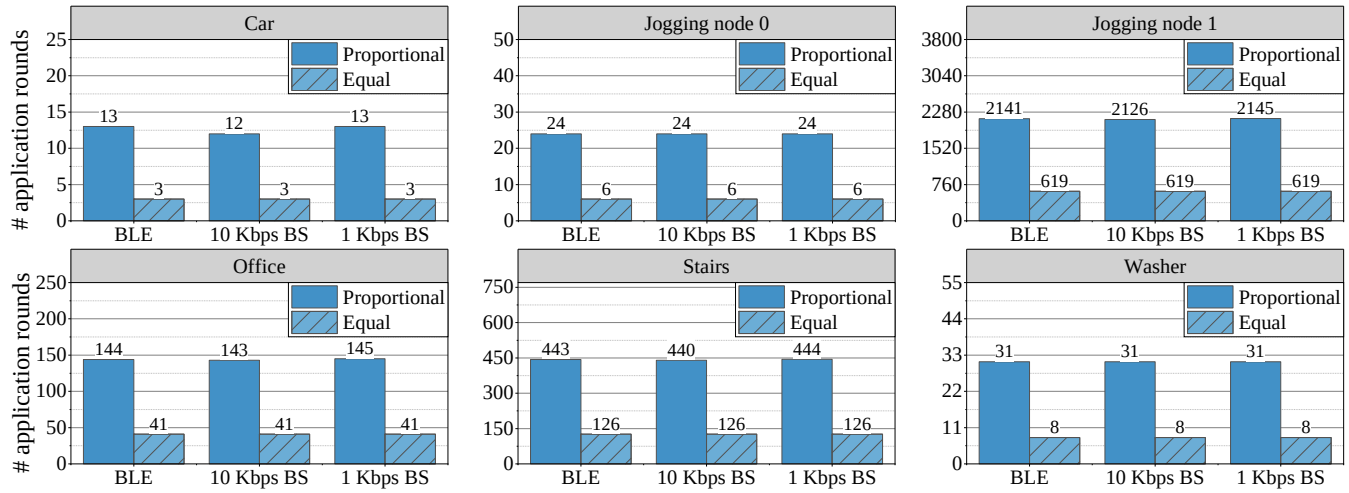


Figure 13: Application rounds for the TADA-based PHM implementation with different charging strategies. *Unlike Fig. 12, this time charging capacitors proportionally favors controlling the camera.*

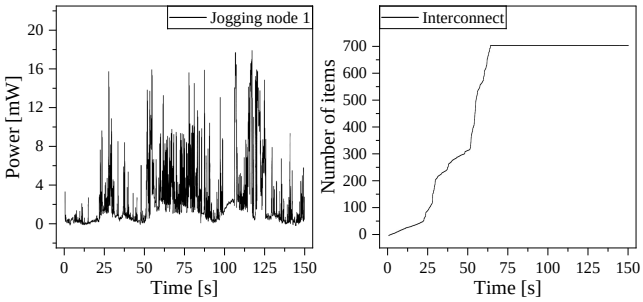


Figure 14: An excerpt of the jogging power trace, compared with the number of items stored in the interconnect for the TADA-based implementation of AEM using 1 Kbps backscatter communication. *The execution stalls because even if MCUs upstream have sufficient energy to produce more data, there is no space in the interconnect. This happens because the last MCU in the application workflow is too slow pulling data from the interconnect.*

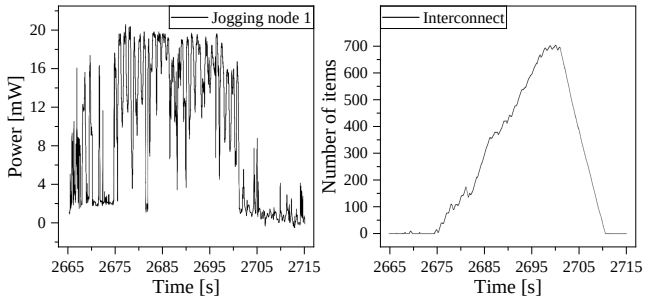


Figure 15: An excerpt of the jogging power trace, compared with the number of items stored in the interconnect for the TADA-based implementation of AEM using 10 Kbps backscatter communication. *Because of a sudden but transient surge in input power, the number of items stored in the interconnect grows up to the limit. The execution stalls until the power burst is over.*

execution of the AEM application using the jogging power trace and 10 Kbps backscatter communication. At time 2675 s, a sudden increase in input power occurs. The MCUs upstream increase their throughput and start pushing data to the interconnect faster than the MCUs downstream can consume. The number of items stored in the interconnect grows until it reaches the limit at time 2698 s. As soon as the burst is over, however, data is progressively pulled from the interconnect, which is eventually back in the same state as before the burst. Note that this situation only happens when TADA is using the equal charging strategy, because it can provide more energy to the smaller capacitors, decreasing their charging time. Once the power burst is over, the MSP430FR5969 MCU used to handle transmissions can progressively pull the data from the interconnect until it is eventually empty again.

6 DISCUSSION

The results we gather in Sec. 5 are a function of two key aspects: the selection of MCUs around the interconnect and the choice of benchmark applications. We discuss how these aspects possibly impact the conclusions we draw.

MCUs. The MCUs we use for the application and interconnect prototypes are representative of the major classes of MCUs currently on the market. The MSP430FR5969 MCU is a modern low-power 16-bit MCU providing efficient I/O. The STM32L432 MCU represents higher-power 32-bit MCUs with hardware FPU. The MAX78000 MCU provides hardware acceleration for specific workloads.

Many other MCUs are available that, however, arguably belong to either of the three classes above. For example, other MSP430 MCUs as well as Cortex M0+ MCUs provide performance within the same ballpark figures of the MSP430FR5969 MCU, and not anywhere close to any of the other two classes. Cortex M33 and M7 cores, with their hardware FPUs, belong together with STM32L432

MCUs. Other hardware accelerators, such as the MAX78002 MCU, offer performance way closer to the MAX78000 than to MCUs in the other two classes. Based on this key observation, we maintain that choosing a different combination of MCUs for the evaluation is surely going to change the absolute numbers, yet the key conclusions stay the same.

Benchmark applications. The processing stages of many IoT applications are fundamentally similar [42], including sensing, computing, and transmission. The primary difference is the *size of sensed data* and *processing complexity* [42].

We argue that the three applications we prototype, described in Sec. 4, meaningfully cover the application spectrum. Indeed, we experiment with both large data sizes in PHM and small, yet higher frequency data in HAR. We also consider regular signal processing in AEM and embedded inference in PHM and HAR. The two options for data transmission also allow us to study the effects of different trade-offs between data rates and energy consumption.

Limitations. To run applications on TADA, developers must split possibly monolithic implementations into transactional tasks, which may impose a burden. This process is germane not only to TADA but also to any other task-based programming system [17, 36, 38, 52]. A few tools exist to assist that help developers reduce this burden [18]. These tools can equally support TADA developers.

It is finally worth noting that the TADA is increasingly less beneficial as the energy consumption of peripherals in the system outweighs that of the local processing on the MCUs. This is somehow evident in the PHM application, where an energy-hungry camera dominates the energy consumption. As shown in Fig. 6, TADA does not achieve significant energy savings for the PHM application compared to the other two applications.

7 CONCLUSION

We presented TADA, a system architecture designed to enable efficient execution of IoT applications powered by ambient energy harvesting across heterogeneous MCUs. Programmers partition the application logic by deploying different processing tasks to the most energy-efficient MCU for the job. TADA provides the efficient run-time support enabling decoupled task executions in *time* and *space*. We concretely achieve this through a special-purpose hardware interconnect we design, which enables asynchronous message passing across multiple MCUs with very limited energy overhead. The interconnect also provides persistent storage to cross periods of energy unavailability, offering intermittent computing support to MCUs with no built-in NVM.

We use an MSP430FR5969 MCU to prototype the TADA interconnect and build three staple IoT applications we use as benchmarks. When compared their energy and throughput performance with the single most efficient MCU, our prototype achieves energy savings of up to 96.7% per single execution. Given the same energy budget, this yields a 68.7x throughput improvement.

Acknowledgements. This work is supported by the Swedish Foundation for Strategic Research (SSF) and by the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.3 - Call for tender No. 1561 of 11.10.2022 of the Italian Ministero dell'Università e della Ricerca (MUR); funded by the European Union - NextGenerationEU.

REFERENCES

[1] CC13xx/CC26xx wireless MCUs. https://www.ti.com/lit/ml/szzp146/szzp146.pdf?ts=1719330688992&ref_url=https%253A%252F%252Fwww.ti.com%252Fsitesearch%252Fen-us%252Fdocs%252Funiversalsearch.tsp%253FlangPref%253Den-US.

[2] K32 L Series. https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/k32-l-series-arm-cortex-m4-m0-plus:K32-L-Series.

[3] Keysight power supply E36313A. https://www.keysight.com/hk/en/support/E36313A/160w-triple-output-power-supply-6v-10a-2x-25v-2a.html.

[4] MAX78000. https://www.analog.com/en/products/max78000.html.

[5] MSP430FR5969. https://www.ti.com/product/MSP430FR5969.

[6] STM32 model zoo. https://github.com/STMicroelectronics/stm32ai-modelzoo.

[7] STM32L432KC. https://www.st.com/en/microcontrollers-microprocessors/stm32l432kc.html.

[8] M. Afanasov et al. Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus. In *International Conference on Embedded Networked Sensor Systems (SENSYS)*, 2020.

[9] S. Ahmed et al. The betrayal of constant power× time: Finding the missing joules of transiently-powered computers. In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, 2019.

[10] S. Ahmed et al. Efficient intermittent computing with differential checkpointing. In *International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, 2019.

[11] Saad Ahmed et al. The internet of batteryless things. *CACM*, 2024.

[12] D. Balsamo et al. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters (ESL)*, 2014.

[13] D. Balsamo et al. Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2016.

[14] Ari Y Benbasat and Joseph A Paradiso. A compact modular wireless sensor platform. In *Fourth International Symposium on Information Processing in Sensor Networks (IPSN)*, 2005.

[15] N. Bhatti and L. Mottola. Harvos: Efficient code instrumentation for transiently-powered embedded sensing. In *International Conference on Information Processing in Sensor Networks (IPSN)*, 2017.

[16] N. A. et al. Bhatti. Energy harvesting and wireless transfer in sensor network applications: Concepts and experiences. *ACM Transactions on Sensor Networks*, 2016.

[17] A. Colin et al. Chain: Tasks and channels for reliable intermittent programs. In *Proc. of ACM SIGPLAN Intl. Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2016.

[18] Alexei Colin and Brandon Lucia. Termination checking and task decomposition for task-based intermittent programs. In *Proceedings of the 27th International Conference on Compiler Construction*, 2018.

[19] Alexei Colin, Emily Ruppel, and Brandon Lucia. A reconfigurable energy storage architecture for energy-harvesting devices. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018.

[20] F. Conti et al. Energy-efficient vision on the PULP platform for ultra-low power parallel computing. In *IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE.

[21] S. N. Daskalakis et al. Ambient backscatterers using FM broadcasting for low cost and low power wireless applications. *IEEE Transactions on Microwave Theory and Techniques*, 2017.

[22] B. Denby et al. Kodan: Addressing the computational bottleneck in space. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2023.

[23] Prabal Dutta and David Culler. Epic: An open mote platform for application-driven design. In *International Conference on Information Processing in Sensor Networks (IPSN)*, 2008.

[24] Federico Ferrari, Marco Zimmerling, Luca Mottola, and Lothar Thiele. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 1–14, 2012.

[25] Matthew Furlong et al. Realistic simulation for tiny batteryless sensors. In *Proceedings of the 4th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, 2016.

[26] R. Gajjar et al. Real-time detection and identification of plant leaf diseases using convolutional neural networks on an embedded platform. *The Visual Computer*, 2022.

[27] K. Geissdoerfer and M. Zimmerling. Learning to communicate effectively between battery-free devices. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022.

[28] A. Gholami et al. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*. 2022.

[29] G. Gobieski et al. Intelligence beyond the edge: Inference on intermittent embedded systems. In *Proc. of Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[30] Josiah Hester and Jacob Sorber. Flicker: Rapid prototyping for the batteryless internet-of-things. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, 2017.

[31] Robin Heydon and Nick Hunn. Bluetooth low energy. *CSR Presentation, Bluetooth SIG https://www. bluetooth. org/DocMan/handlers/DownloadDoc. ashx*, 2012.

[32] N. et al. Ikeda. Soil-monitoring sensor powered by temperature difference between air and shallow underground soil. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2020.

[33] J. Ko et al. Low power or high performance? a tradeoff whose time has come (and nearly gone). In *Wireless Sensor Networks: 9th European Conference (EWSN)*, 2012.

[34] Konstantinos Koukos et al. Multiversioned decoupled access-execute: The key to energy-efficient compilation of general-purpose programs. In *Proceedings of the 25th International Conference on Compiler Construction*, 2016.

[35] V. Liu et al. Ambient backscatter: Wireless communication out of thin air. *ACM SIGCOMM computer communication review*, 2013.

[36] B. Lucia et al. A simpler, safer programming and execution model for intermittent systems. In *Proc. of ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*, 2015.

[37] K. Ma et al. Architecture exploration for ambient energy harvesting nonvolatile processors. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[38] K. Maeng et al. Alpaca: Intermittent execution without checkpoints. *Proc. of ACM SIGPLAN Intl. Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 1, 2017.

[39] K. Maeng and B. Lucia. Adaptive dynamic checkpointing for safe efficient intermittent computing. In *International Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[40] A. Maioli et al. Discovering the hidden anomalies of intermittent computing. In *International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2021.

[41] A. Maioli and L. Mottola. Alfred: Virtual memory for intermittent computing. In *International Conference on Embedded Networked Sensor Systems (SENSYS)*, 2021.

[42] J. Mocnej et al. *Network traffic characteristics of the IoT application use cases*. School of Engineering and Computer Science, Victoria University of …, 2018.

[43] B. O'Flynn et al. The development of a novel minaturized modular platform for wireless sensor networks. In *Fourth International Symposium on Information Processing in Sensor Networks (IPSN)*, 2005.

[44] J. Portilla et al. A modular architecture for nodes in wireless sensor networks. *J. Univers. Comput. Sci.*, 2006.

[45] B. Ransford, J. Sorber, and K. Fu. Mementos: System support for long-running computation on rfid-scale devices. In *International Conference on Architectural Support for Programming Languages and Operating systems (ASPLOS)*, 2011.

[46] James E Smith. Decoupled access/execute computer architectures. *ACM SIGARCH Computer Architecture News*, 1982.

[47] F. Sutton et al. Bolt: A stateful processor interconnect. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, 2015.

[48] M. et al. Thielen. Human body heat for powering wearable devices: From thermal energy to application. *Energy conversion and management*, 2017.

[49] N. Van Huynh et al. Ambient backscatter communications: A contemporary survey. *IEEE Communications surveys & tutorials*, 2018.

[50] J. Woude and M. Hicks. Intermittent computation without hardware support or programmer intervention. In *Symposium on Operating Systems Design and Implementation (ODSI)*, 2016.

[51] Yang Xu and Ting Ting Qiu. Human activity recognition and embedded application based on convolutional neural network. *Journal of Artificial Intelligence and Technology*, 2021.

[52] Kasım Sinan Yıldırım et al. Ink: Reactive kernel for tiny batteryless sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, 2018.

[53] Xian-Da Zhang. *Modern signal processing*. Walter de Gruyter GmbH & Co KG, 2022.