Near-Optimal Multi-Accelerator Architectures for Predictive Maintenance at the Edge

Mostafa Koraei[a], Juan M. Cebrian[b], Magnus Jahre[c]

[a]*SINTEF AS, Oslo, Norway*
[b]*University of Murcia, Murcia, Spain*
[c]*Norwegian University of Science and Technology (NTNU), Trondheim, Norway*

Abstract

Predictive maintenance systems face a rich set of constraints along dimensions such as latency, throughput, physical size, monetary cost, as well as energy and power consumption.

To meet performance requirements, predictive maintenance systems require specialized compute units (i.e., accelerators) in addition to conventional processor cores. Unfortunately, size and cost constraints commonly result in developers being forced into selecting System-on-Chip (SoC) platforms that do not have sufficient resources to fully accelerate all performance-critical functions — in essence raising the challenging question of how to optimally distribute the available resources across accelerators. This work introduces the *Resource-Constrained Accelerator Selection (RCS)* methodology, which identifies near-optimal multi-accelerator configurations for predictive maintenance applications.

RCS takes a library of resource-scalable accelerator architectures as input and then selects the combination of accelerator configurations that minimizes end-to-end latency. We find that enabling RCS for typical predictive maintenance applications requires a resource-scalable Fast Fourier Transform (FFT) accelerator and propose *ScaleFFT* to fill this gap.

We apply RCS and ScaleFFT to a collection of edge computing applications with different sensor bandwidths and find that they reduce end-to-end latency by 2.4× on average for a 256K-point FFT compared to a state-of-the-art configuration that only accelerates the machine learning algorithm. Moreover, we demonstrate that RCS enables real-world gains in oil well and train track monitoring systems.

Keywords: Edge computing, FPGA, Accelerator, FFT, Predictive maintenance, Machine learning
2010 MSC: 00-01, 99-00

1. Introduction

Edge computing systems move a significant part of a system's computational load from the cloud-based back-end to the devices that capture data from, and potentially act upon, the environment. The motivation for edge computing is (i) to improve response time by moving critical decisions from the cloud to the edge, (ii) avoid overwhelming the network or cloud-based back-end with a massive amount of data, or (iii) leverage the distributed nature of the system to make decisions that benefit from both local and global perspectives [1]. For example, many systems have to perform inference locally as the network links lack the bandwidth to transmit all sensor data [2]. An interesting application area are the so-called Industry 4.0 applications, which introduce devices that continuously collect data about the status of production assets. This enables *Predictive Maintenance (PdM)* which provides an opportunity to reduce operating costs by minimizing unplanned downtime and avoiding equipment failure [3].

In general, PdM systems perform three main functions. First, they periodically or continuously monitor an asset, e.g., using accelerometers or ultrasonic sensors to sense vibrations. Second, they perform conventional signal processing on the captured data, such as a Fast Fourier Transform (FFT), to make it more amendable to analysis, e.g., converting the signal from the time domain to the frequency domain. Third, they apply a machine learning algorithm to detect anomalies. Differentiating normal and abnormal behavior in the asset is typically much simpler in the frequency domain. For example, an issue with the train track will typically manifest itself as vibrations at abnormal frequencies in the bogie of the train, and this is more easily detected in the frequency domain than in the time domain.

Designing PdM edge systems is challenging because they are heavily constrained in terms of latency, throughput, power dissipation (e.g., heat), energy consumption (e.g., battery lifetime and capacity), physical size, and cost. Moreover, many industrial applications are produced in low volume, i.e., as little as tens or hundreds of systems are sufficient to meet demand. Creating custom Application Specific Integrated Circuits (ASICs) is hence not economically viable due to high non-recurring engineering costs [4]. At the same time, accelerating signal processing and machine learning algorithms is typically critical to meet latency and energy constraints.

Developers are hence forced to select an off-the-shelf Sys-

*Juan M. Cebrian
Email addresses: Mostafa.Koraei@sintef.no (Mostafa Koraei), jcebrian@um.es (Juan M. Cebrian), magnus.jahre@ntnu.no (Magnus Jahre)

tem on Chip (SoC) platform that (i) supports acceleration, (ii) interfaces cleanly with the application-specific sensors, and (iii) contains conventional processor cores on which the application parts that are non-acceleratable can be executed. The key alternatives are SoCs that combine processor cores with a Field Programmable Gate Array (FPGA) (e.g., Xilinx Zynq [5] or Intel Agilex [6]), a Graphics Processing Unit (GPU) (e.g., NVIDIA's Tegra [7]), or a Digital Signal Processor (DSP) (e.g., Texas Instrument's KeyStone Architecture [8]); industrial PCs such as Intel NUC [9] have insufficient support for accelerators. Real-world PdM systems tend to favor FPGA-enabled SoCs because they are significantly easier to interface with sensors than GPU- and DSP-based SoCs. For example, the intricacies of interfacing with sensors resulted in that the DSP-based 66AK2G1x KeyStone SoC could only read sensor data at a rate of 64 Kilo-Samples Per Second (KSPS), while the FPGA-based Zynq SoC easily reached the 128 KSPS and 256 KSPS required for our train and oil well use cases, respectively. PdM systems hence typically require an FPGA to efficiently interface with sensors. In addition, using the FPGA for acceleration reduces the number of components in the system and therefore its cost and physical size. FPGAs are however challenging to program [10], and it is hence attractive to leverage reusable accelerator architectures. FPGA-vendors refer to such reusable accelerators as Intellectual Property (IP) cores.

Our goal in this paper is to understand how best to select and size accelerators for PdM at the edge. If the constraints are sufficiently relaxed, the issue is trivial as one can simply size each accelerator maximally, i.e., combine a fully accelerated FFT with the best-performing machine learning accelerator configuration. The problem becomes more challenging when resources are limited — essentially posing the question of how much of the limited FPGA resources should be devoted to each accelerator. We observe that answering this question requires two key technologies. More specifically, we need (i) a methodology for selecting the accelerator configurations that collectively minimize end-to-end latency under resource constraints, and (ii) resource-scalable accelerators, i.e., accelerators that can be configured to occupy different design points in terms of FPGA resource consumption and performance. Prior works that focus on accelerator selection methodologies [11, 12, 13] fall short because they do not consider resource-scalable accelerators. In addition, we find that resource-scalable neural network accelerators are abundant (e.g., [14, 15, 16]), but state-of-the-art FFT accelerators (e.g, [17, 18, 19, 20]) are not resource-scalable because they assume that the system has sufficient resources to fully accelerate the FFT.

Our first key contribution is hence the *Resource-Constrained Accelerator Selection (RCS)* methodology. RCS takes a library of accelerator families with configurations that span the performance versus resource consumption spectrum of each family as input. RCS then uses the information about resource consumption and latency available in the library to select the configuration of each accelerator that minimizes the predicted end-to-end latency. For PdM systems, end-to-end latency is the time it takes from the sensor starts capturing the first sample until all data is fully processed. End-to-end latency is the key performance metric for PdM systems because they repeatedly (i) capture sample(s), (ii) use an FFT to transform sample data from the time domain to the frequency domain, and (iii) apply machine-learning on the frequency-domain representation to detect abnormal behavior. Depending on the application, the PdM system is either stationary or mobile. In a stationary system, the end-to-end latency determines time-on-station, i.e., the amount of time the system must remain stationary to inspect the asset (e.g., our oil well use case). In a mobile system, the end-to-end latency determines spatial resolution, i.e., the amount of time the system will move before new samples can be collected (e.g., our train track monitoring use case).

While resource-scalable machine learning accelerators are abundant[1], no resource-scalable FFT accelerator exists. Our second key contribution is hence a resource-scalable FFT accelerator architecture which we call *ScaleFFT*. Our key insight is that we can create a resource-scalable accelerator family by exploiting the hierarchical nature of the FFT transform. More specifically, all ScaleFFT configurations consist of two accelerators, one for the Discrete Fourier Transform (DFT) and one which reorganizes data in a butterfly pattern. ScaleFFT then uses the conventional processor cores, which edge PdM systems anyway require to implement non-performance-critical functionality, to manage the two accelerators. In this way, ScaleFFT is able to create accelerator configurations that are positioned between fully accelerated configurations such as Xilinx' XFFT [22] and accelerators proposed in prior work [17, 18] — which are fast but require significant FPGA resources — and a software-only FFT implementation — which is slow because it runs on a processor core but requires no FPGA resources.

We evaluate RCS and ScaleFFT on a representative FPGA-accelerated edge computing platform running PdM applications ranging from medium to high signal bandwidth. More specifically, we consider 32, 64, and 256 KSPS configurations that are representative of PdM applications that monitor a range of assets, including wind turbine gear boxes, train bogies, and acoustic/vibration inspections for oil wells. Moreover, we consider CNNs with different computational requirements for each signal bandwidth. RCS identifies the optimal configuration point in all applications we consider, i.e., RCS selects the ScaleFFT and Xilinx DPU configurations that use the FPGA resources most efficiently. That said, RCS is heuristics-based so it is in general difficult or even impossible to guarantee optimally. RCS reduces end-to-end latency by 2.4× on average for a 256K-point FFT compared a state-of-the-art approach that solely accelerates the CNN while satisfying all constraints.

To investigate the effects of using RCS in real-world PdM applications, we applied it to an oil well inspection system and a railway track monitoring system. Oil wells must be periodically inspected for safety; production is stopped during inspection. The speed at which the oil well can be inspected is hence critically important as the facility is essentially idle during this operation. We must also consider the additional expenses due to the operation time of a drilling platform or drill ship, which can

[1] RCS uses Xilinx' Deep Learning Processing Unit (DPU) IP [21] as the Convolutional Neural Network (CNN) accelerator family in this work.

be as high as 200K to 400K USD per day, yielding 12.5K USD per hour on average [23]. RCS reduces inspection time from 2.1 hours to 1.7 hours (for 120 points of inspection), thereby saving an estimated 5.0K USD compared to our baseline (FPGA accelerated DPU, software FFT) for each inspection job. The inspection system we consider in this work is scheduled to be deployed at several oil installations in the upcoming years.

Railway infrastructures play a critical role in transportation and logistics and must be reliable. It is hence attractive to continuously monitor train tracks by installing a monitoring system in train bogies to detect problems early and thereby avoid service disruption. We evaluate RCS in the context of the TrainDAQ [24] track monitoring system. TrainDAQ uses high-bandwidth accelerometers (i.e., 15 to 30 kHz) and high-speed and high-resolution Analog to Digital Converters (ADCs) to monitor the train track and feeds the signal to an FPGA-accelerated edge computing platform to perform signal processing and machine learning inference and thereby identify anomalies. In TrainDAQ, higher performance results in better spatial resolution, i.e., each train will inspect more of the track, and we find that RCS improves spatial resolution by 54% compared to the baseline by better utilizing the FPGA.

2. The Anatomy of Predictive Maintenance Systems

Broadly speaking, there are three main maintenance management strategies. The *run-to-failure strategy* performs maintenance only after the occurrence of failures. This simple approach is frequently adopted, but it has significant costs associated with downtime after failure. The *preventive maintenance strategy* schedules actions based on vendor statistics on component failure rate. This statistical data is usually conservative which means that components are replaced way before they will fail — leading to inefficient use of resources and increased costs. The *Predictive Maintenance (PdM) strategy* is hence a best-of-both-worlds approach in which PdM systems monitors key components to assess their current condition [3]. In other words, PdM aims to detect when a component needs to be replaced which hence reduces costs compared to preventative maintenance — because components are replaced when necessary rather than conservatively — and reduces downtime compared to run-to-failure — because components are replaced before they fail.

Figure 1 shows a generic PdM system that leverages an FPGA-based SoC with a sampling process that consists of three key steps. The first step is to capture sensor data (see ❶). Real-world PdM systems typically need a large number of sensors (e.g., the train bogie PdM system that we will discuss in detail in Section 7 has 16 sensors). The amount of sensor data is too much to be stored on-chip and it is hence written to main memory through the Direct Memory Access (DMA) controller (see ❷). The second step is to transform sensor data from the time domain to the frequency domain with the FFT accelerator (see ❸). The FFT accelerator reads the sensor data from main memory, computes the FFT, and writes the output back into main memory. The third step is to perform machine learning inference on the frequency-domain representation of the sensor data (see ❹). This is done using the Deep Learning Processing Unit (DPU) and leverages a failure prediction model that has been trained offline. A logical overview of stages ❸ and ❹ can be seen in Figure 2. Pattern selection is performed by the CPU. The output of DPU processing is either that the component is operating normally or that a problem has been detected (see ❺). If a problem has been detected, the system notifies an operator and a maintenance action is scheduled.

The above approach is a good match for various PdM applications as it performs all analysis locally.

This is a critical requirement because many real-world PdM deployments either do not have network connectivity or the network does not have sufficient capacity to transfer raw data. A typical PdM use case is to use vibration sensors to detect bearing wear, for instance in metal lathe [25], slitting machines [26], vehicle factories [27], woodworking [28], compressors [29], and other industrial equipment [30, 31, 32, 33, 34, 35], but adding the network infrastructure for transferring raw vibration data is impractical or even impossible in such industrial environments. In other cases, for instance wind turbines [36, 37] and the electrical network [38, 39, 40, 41], the data could be stored on the device and downloaded manually when an operator inspects the installation. In this case, the challenge is that maintenance schedules can vary significantly between operating companies which means that significant storage capacity would need to be added to account for the foreseen worst-case inspection interval. Finally, the risks incurred by relying on offline analysis are typically not acceptable in cases where health and safety is at stake, for example in aeronautics [42, 43, 44, 45], railways [46, 47, 48], fuel cell status in electric vehicles [49], and engine and gearbox analysis [37, 50]. Overall, we analyzed more than 50 PdM approaches referenced in recent surveys [51, 52], and found that most described systems use either cloud or offline resources to analyze sensor data. These approaches hence requires adopting fully local processing, as we describe in this work, to be practically applicable.

3. Resource-Constrained Accelerator Selection

We now describe our RCS approach for configuring a near-optimal multi-accelerator architecture for resource-constrained PdM systems, i.e., how it identifies the accelerator configurations that yield the lowest end-to-end latency within the set of accelerator configurations that satisfy all constraints. For each accelerator type i (e.g., ScaleFFT), the accelerator vendor synthesizes netlists at different performance versus resource-consumption design points to yield a set of possible accelerator configurations A_i. The benefit of doing this is twofold. First, this means that the time overhead of synthesizing the accelerators is incurred once for each accelerator variant. Second, it means that the resource consumption (e.g., number of DSP blocks, block RAMs, lookup tables, and flip-flops) and latency of each accelerator configuration can be accurately predicted. Our proposed approach is in line with current industry practice. More specifically, Xilinx ships a collection of pre-synthesized configurations for the DPU we use this work [21]. This step is hence a one-time cost for each accelerator type.

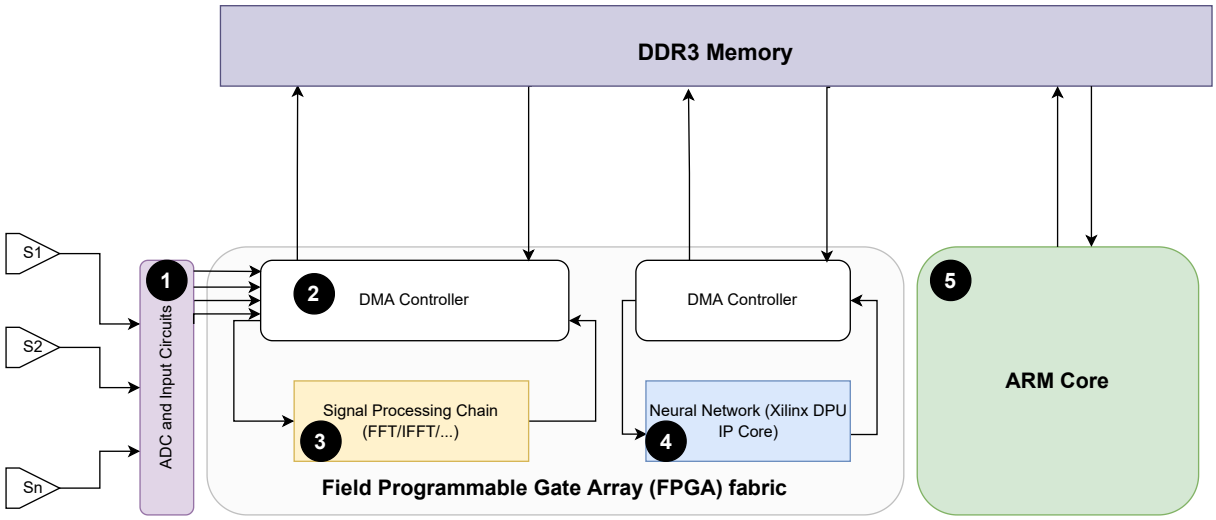

Figure 1: Architecture of an FPGA-based PdM system. *The FPGA is responsible for (i) interfacing to the Analog-to-Digital Converters (ADCs) to write sample data to memory, (ii) transforming sensor data into the frequency domain by applying the FFT, and (iii) detect failures by performing machine learning inference. The ARM cores in the SoC executes non-accelerated functions and manages the accelerators.*

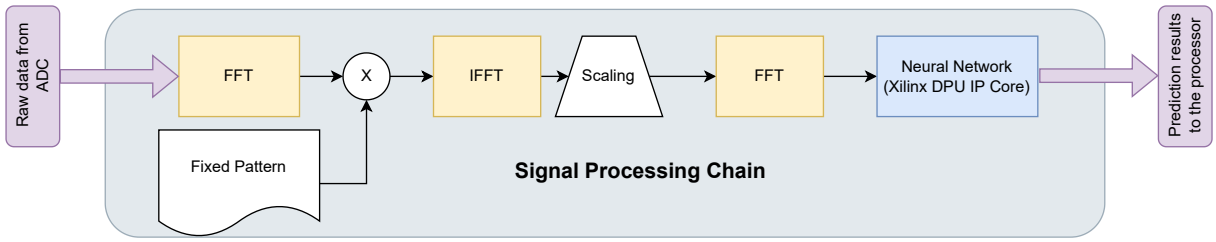

Figure 2: Overview of the signal processing chain from raw sensor data to DPU inference.

The next step is to predict end-to-end latency T across accelerator configurations, and we use a simple performance model to achieve this. More specifically, we assume that the application consists of a part that cannot be accelerated ($t_{\text{not-accelerated}}$) as well as n tasks that can be accelerated:

$$T = t_{\text{not-accelerated}} + \sum_{i=0}^{n} t_i^{a \in A_i} \quad (1)$$

Each task i takes t_i^a seconds to execute when using accelerator configuration a, and the accelerator configurations that can be used to accelerate task i are of type i and hence part of the set A_i, e.g., ScaleFFT can accelerate the FFT task but not machine learning inference. Recall that t_i^a can be retrieved directly from the vendor-provided synthesis results. We further include the CPU execution time of each task in A_i as an accelerator which uses no resources. For applications that can be pipelined, we set t_i^a to the latency of producing all results for the last pipeline stage and the latency of producing the first result for the pipeline stages that overlap with the last pipeline stage.

In addition to guiding accelerator selection, Equation 1 also yields insight into when acceleration is an effective optimization. More specifically, $t_{\text{not-accelerated}}$ bounds the attainable speed-up according to Amdahl's law [53], i.e., end-to-end latency will converge towards $t_{\text{not-accelerated}}$ as more performant accelerator configurations are selected. Acceleration is thus most favorable when $t_{\text{not-accelerated}}$ accounts for an insignificant part of end-to-end latency. For our benchmarks, $t_{\text{not-accelerated}}$ is the time it takes to collect samples, which accounts for maximally 18% of end-to-end latency in our baseline. Similarly, accelerator selection is trivial when the largest accelerator configuration of each type can be selected while satisfying resource constraints.

We focus on PdM systems where system requirements (e.g., cost or physical size) forces developers to select platforms in which maximally sizing all accelerators require more resources than are available. The next step in the RCS methodology is hence to select resource-feasible accelerator configurations which yield near-optimal end-to-end latency. We start by identifying all accelerator variants that (i) can be used by the target application, and (ii) do not require more of any resource than those available in the target device. We then evaluate all combinations of accelerator types. If the resource consumption of the combination of accelerators is less than U% of any resource, we predict end-to-end latency using Equation 1. The reason for not drawing the limit at 100% utilization is that the post-synthesis mapper typically has to time-multiplex different components onto the same resource to fit the design onto the FPGA when a resource is nearly fully utilized, resulting in a latency overhead. We empirically determined that setting U to 95% is sufficient to avoid this issue.

There are two key reasons why the accelerator configura-

tion selected by RCS can differ from the optimal accelerator configuration. First, the accelerator (IP-core) vendor only provides a discrete set of accelerator configurations, and it is hence possible that the target PdM system would perform better with an accelerator configuration that is not in this set. The number of feasible configurations of an accelerator is however typically limited and it is hence unlikely that vendors do not pre-synthesize attractive configurations. Second, it is not guaranteed that the accelerator will achieve its predicted latency. Our evaluation in Section 6 however shows that our aforementioned utilization-limiting heuristic is effective.[2] For these reasons, we claim that RCS is able to select near-optimal accelerator configurations for resource-constrained PdM systems.

RCS evaluates $O(m^n)$ configurations in the worst case where m is the maximum number of configurations for any accelerator type and n is the number of accelerator types. While the run-time of RCS would be a concern if the number of accelerator types and configurations were large, resource-constrained PdM systems have few accelerator types and configurations. More specifically, we consider two accelerator types, the DPU and ScaleFFT, with 2 and 6 configurations, respectively, while contemporary workstations can easily evaluate many millions of data points in seconds. Moreover, (i) the search can be easily parallelized, and (ii) it is not necessary to further explore configurations that exceed the resource constraint after considering a subset of accelerator types. Since search time is not an issue for resource-constrained PdM systems, we leave the exploration of better search heuristics for future work.

4. ScaleFFT: Resource-Scalable FFT Acceleration

As we have seen, FFT transformations are a critical step in many predictive maintenance systems. The faster we move the data from time to frequency domain, the sooner we can feed the data to the corresponding machine learning algorithms and in turn enable early detection of system malfunctions. FFT transformations are however computationally expensive, and therefore typically requires hardware acceleration to satisfy performance requirements.

4.1. State-of-the-art FFT Accelerators

There are several strategies for accelerating FFT transformations. One approach is to use embedded vector processors or low power GPUs, but these architectures are usually expensive and have energy requirements that are not feasible for continuous monitoring purposes. Digital signal processors (DSPs) are a much better option regarding price and energy, but they usually have a fixed FFT width, making it harder to adapt to different scenarios. Moreover, DSPs can be easily overwhelmed when receiving information from many sensors.

[2] This is in part due to our focus on resource-constrained systems. For high-performance FPGA platforms, performance often saturates due to limited memory bandwidth and significant complexity is hence commonly devoted to increase the operational intensity of accelerators (see e.g., [54]). Our platform would saturate memory bandwidth at 512 KSPS which is beyond the requirements of current and emerging PdM systems (e.g., our oil well use case is in the high end of commercially deployed systems and requires 256 KSPS).

Cooley and Tukey laid the foundations of the FFT back in 1965 [55]. Over the course of 50 years, there have been many pipelined FFT hardware proposals [56]. Most proposals share a common idea, to divide the FFT transformation into several discrete Fourier transformations (DFT) that run in parallel, coupled with a butterfly network to produce an identical output (e.g., [17, 18]). Figure 3 shows how an 8-point FFT can be computed by using smaller DFTs based on Cooley's design [55]. More specifically, four 2-point DFTs run in parallel and feed data to two 4-point butterfly networks and then on to a single 8-point butterfly network.

This FFT implementation requires partitioning the input array into multiple sub-arrays that match the input size of the DFTs. For example, a 1024-point FFT using 64-point DFTs would need to process 16 sub-arrays. It is also important to note that sub-array ordering is not linear. Figure 3 also shows the order of the sub-arrays when using 4-point DFTs to generate an 8-point FFT. The first sub array contains following elements 0,2,4,6 and the second one contains 1,3,5,7. If we use 2-point FFTs, sub-arrays are [0,2][4,6][1,3][5,7]. The reordering pattern is specific to an FFT, i.e., it varies across FFTs.

There are many practical constraints that can limit the size of the FPGA, including costs, area and energy. Therefore, and for the sake of example, let us assume that the resources required for performing two 4-point FFT in parallel and the 8-point butterfly network exceed the resources available on the FPGA (faded out in Figure 3). In all previous works, this lack of resources would translate into moving the FFT transformation to a pure software implementation, hence (dramatically) under-utilizing the FPGA.

4.2. Resource-Scalable FFT Acceleration

Our solution, which we call *ScaleFFT* and illustrate in Figure 4, is to design a re-programmable butterfly network that can be used on all stages of the process to emulate the behavior of a bigger FFT implementation via iterative process (e.g., a 4-point butterfly that emulates a 8-point as shown in Figure 4). In this way, we can create FFT accelerators in which performance scales with the amount of FPGA resources allocated — thereby enabling FFT acceleration in resource-constrained predictive maintenance systems which have some FPGA resources available but not enough resources to fully accelerate the FFT.

For the sake of simplicity, we rely on standard intellectual property (IP) cores for implementing FFT accelerators on the FPGA. Indeed, the standard IP library for FFT in Vivado (Xilinx) offers different input lengths for the DFTs. We implement a parallel butterfly network to do the rest of the computations.

Figure 5 provides an overview of ScaleFFT when mapped onto an FPGA-accelerated SoC. Both the FFT-core and the butterfly network have access to the platform's DDR3 memory through an AXI DMA controller. This allows for direct reading and storing of data in memory. A software program running on the ARM-cores of the SoC implements a state machine that controls the data elements to be fed to the FFT-core and butterfly network. A memory pointer generator function is used to manage sub-arrays for DFTs, butterfly network and el-

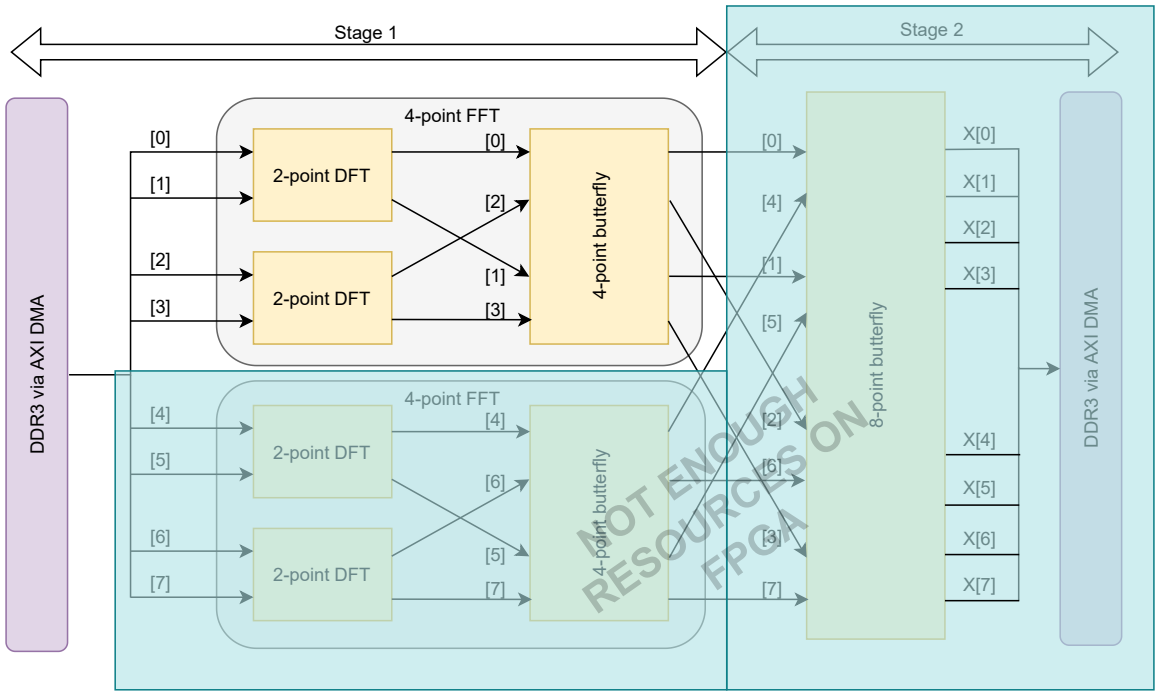

Figure 3: High-level architecture of a state-of-the-art FFT accelerator. *Existing accelerators assume that there are sufficient FPGA resources to fully accelerate the FFT and hence fall short in resource-constrained use cases.*

Table 1: SoC resource comparison.

SoC Model/ Resource	Z-7020	Z-7035
Logic Cells (K)	85	275
Block RAM (Mb)	4.9	17.6
DSP Slices	220	900

ement arrangement. This function also controls the AXI DMA-controller based on the state machine.

5. Experimental Setup

5.1. Platform

We have used a Zynq 7020 and 7035 FPGA modules with a TE0701-06 as carrier board (Table 1). The FPGA of the modules can cover up to 8 kilo-point FFTs based on the Xilinx FFT IP core. We have used an FPGA mezzanine card (FMC) plugged to the carrier board to connect the FPGA to another board with analog to digital converters (ADCs) and analogue front end circuits for the accelerometers. Accelerometers have an integrated electronics piezo-electric (IEPE) interface which converts the high impedance signal of the piezoelectric material into a voltage signal with a low impedance of typically 100 Ω. Most IEPE sensors work at a constant current between 2 and 20 mA. Since our system is usually installed near the sensors we use a 5 mA constant current to reduce the power dissipation. The system is standalone and during benchmarking and monitoring, the system is connected to a PC via an UART interface. For measuring the power dissipation we monitor the current of the FPGA module separated from ADCs and the carrier board. We repeat each test twice and present average clock ticks as reported by performance counters attached to the ARM cores. We measure clock ticks separately for the FFT and neural network inference tasks to report time spent on each function.

5.2. Benchmarks

As discussed in Section 2, predictive maintenance systems typically require a neural network accelerator and an FFT accelerator. We use an IP from Xilinx, the Deep Learning Processing Unit (DPU), as our neural network accelerator. To test our RCS proposal, we use different combinations of DPU implementations and FFT sizes. Each DPU implementation has different FPGA resource requirements, mainly block RAMs (BRAM), registers, DSPs and Look-Up Tables (LUTs). This sets a constraint on the available resources for the FFT transformation. We use two different types of DPU, and for each DPU we evaluate two implementations.

Predictive maintenance applications primarily differ on the frequency bandwidth of the sensors, requiring FFTs with different lengths. We hence choose FFT-lengths that are representative of predictive maintenance applications positioned at different design points such that we span the design space. In wind turbine applications, gearboxes have relatively low speeds. This translates to bandwidth requirements between 5 and 10 KHz for its sensors, and a sample rate of 32 Kilo Samples Per Second

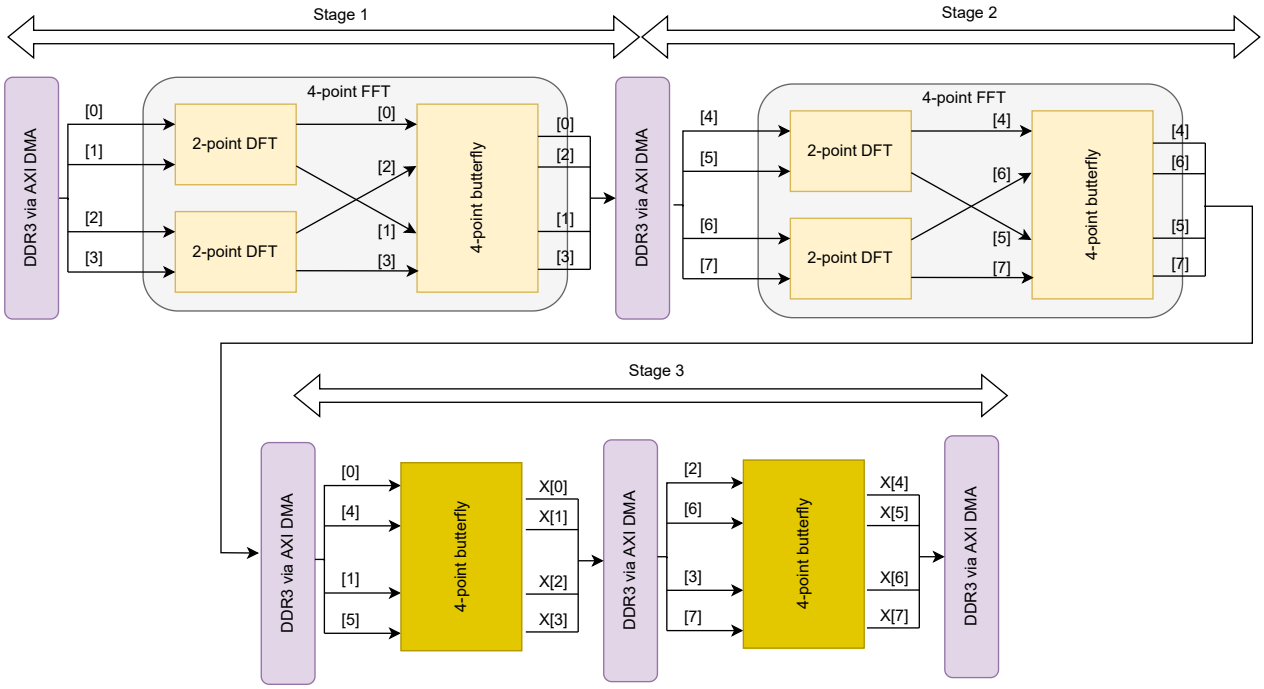

Figure 4: ScaleFFT transforms the FFT into an iterative process of shorter FFTs and butterfly networks. *The resource requirements of each stage can hence be traded off against the number of stages, i.e., providing more resources reduces latency.*

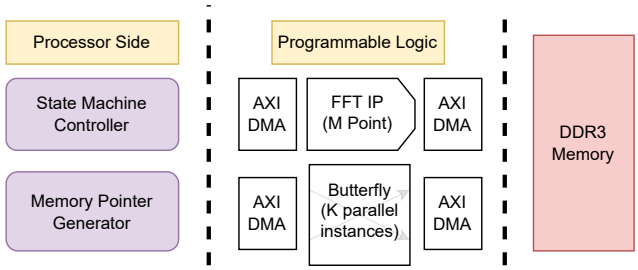

Figure 5: High-level architecture of ScaleFFT mapped to an FPGA-based SoC.

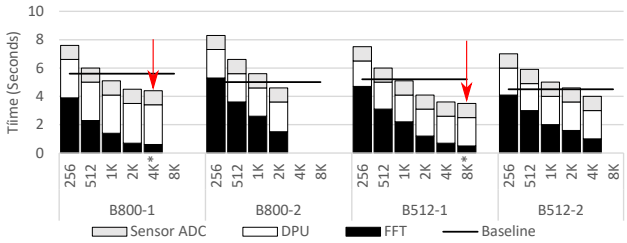

Figure 6: Execution time breakdown for 32K samples FFT on Zynq 7020

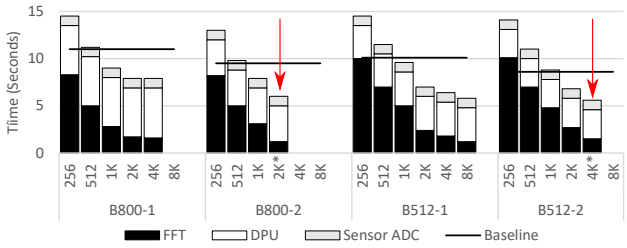

Figure 7: Execution time breakdown for 64K samples FFT on Zynq 7020

(KSPS) is enough to capture all fundamental harmonic frequencies. On the other hand, in train bogie vibration analysis, the frequencies of interest are directly related to the speed of the train, with sensors measuring from 10 to 20 KHz. A sample rate of 64 to 128 KSPS is hence enough for this application. Finally for down-hole oil well inspection, sensor bandwidth's are between 40 and 60 KHz. The recommended sample rate in this scenario is hence between 128 and 256 KSPS. Please note that none of mentioned sampling rates can be fully accelerated on our SoCs due to lack of resources.

6. Results

This section shows the evaluation results in terms of performance, energy and FPGA resource utilization. Our baseline, depicted as a horizontal red line in the figures, uses a software implementation of the FFT and a FPGA-accelerated DPU. BXXX-Y describe combinations of DPU and FFT accelerators. XXX is the DPU parallelism architecture, while Y relates to different implementations of said DPU. DPU B512 has 8 input/output channel parallelism and B800 has 10 input/output channel parallelism. Arrows in the figures show the combination selected by RCS (as described in Section 3), and the true optimal configuration is marked with a star (*). We identify the true optimal configuration by executing all configurations on the target platform and selecting the configuration with the

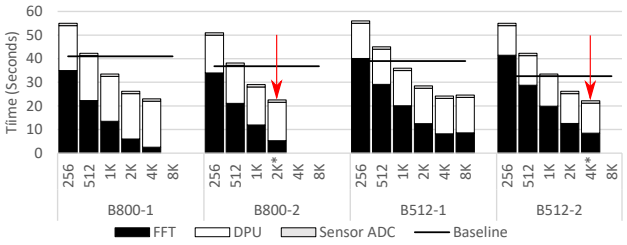

Figure 8: Execution time breakdown for 256K samples FFT on Zynq 7020

lowest end-to-end latency; recall that RCS predicts end-to-end latency from vendor-provided timing information.

6.1. Performance

Figures 6, 7, and 8 show end-to-end latency broken down into sensing, FFT, and DPU computations for different FFT sampling sizes. These sampling sizes match the applications discussed in Section 5.2. The x-axis shows the actual size of the FFT accelerator, ranging from 256-point FFT to the maximum size allowed by the available resources, between 2K and 8K, depending on the DPU/FFT combination. All configurations show good scalability until a saturation point, outperforming the baseline design by a significant amount as sampling size increases. For example, for a 256K-point FFT, the configuration selected by RCS speeds up execution from 47% to 78%, depending on the selected DPU.

There are two key scalability limiting factors for our proposed FFT accelerator. The first one has to do with FPGA synthesis optimization based on resource utilization. Indeed, Figure 9 shows the resource utilization for the B512-1 design for the 256K-point FFT. For a 8K FFT implementation, we are reaching 98% BRAM usage, meaning that the synthesis tool will have less chances to find a performance-optimal design. Since we are over our threshold value of 95% (see Section 4), the 8K implementation is not selected by RCS, but the 4K (Figure 8). This synthesis variability translates into fluctuations in the performance scalability as we double the resources allocated to the FFT accelerator. This can be seen most clearly for B512-1 regardless of input size, as it slowly saturates after the 4K implementation.

On the other hand we have the utilization of FFT and butterfly network themselves. For example, B800-1 shows good scalability up to 2K length for 32K and 64K-point FFTs, with 25% and 60% speedups, respectively. However, for 256K-point FFTs, it scales with 4K too (78% speedup). This is due to the fact that, for relatively small inputs, the main bottleneck of our design is the FFT-core (Figure 4). As the input size increases, the pressure on the butterfly network does too. For example, a 256K-point FFT needs to run 2 additional butterfly passes on all points. If we compare a 64K-point FFT implementation using a 4K FFT IP to a 256K-point FFT, we need double the butterfly network times. Since more time is spent on the butterfly network, and it is not currently a bottleneck, we see further scalability with accelerator size.

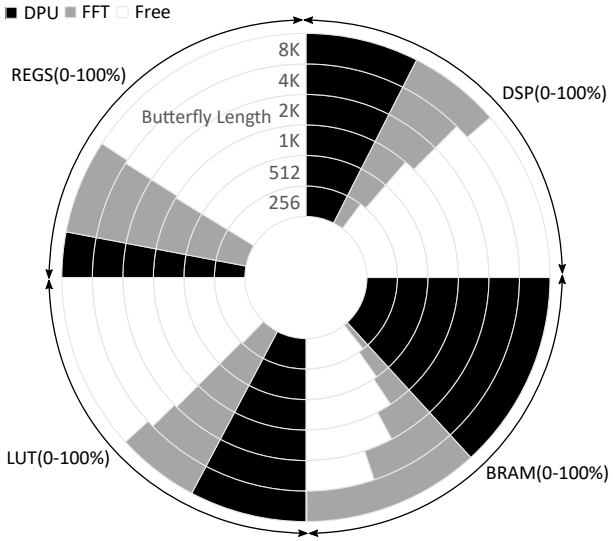

Figure 9: Resource breakdown for B512-1 on Zynq 7020 (256K-point FFT)

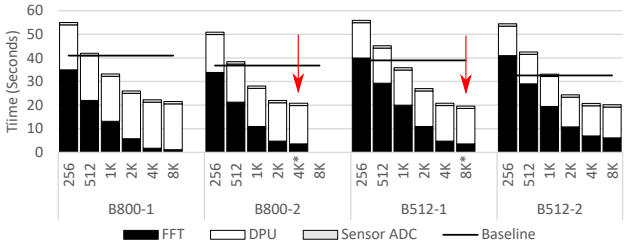

Figure 10: Execution time breakdown for 256K samples FFT on Z7035 FPGA

We see the opposite behavior for B512-1 and the 256K point FFT. In this case, the extra load on the butterfly network combined with the near complete usage of BRAMs (98%, as shown in Figure 9) provokes a slowdown in performance. To show that this is resource-related, we synthesized our designs in a Z-7035 SoC, depicted in Figure 10. This figure shows a slight scalability improvement for B512-1, and, since resources are below the 95% threshold, the 8K implementation is selected by RCS.

6.2. Power/Energy Efficiency

We measured the FFT FPGA implementation power dissipation in the range of 150 to 250 mW in real hardware. These values match the estimations from Xilinx XPE 2019 tool that reports 200 to 300 mW for the FPGA-side. FPGA power dissipation is relatively small compared to system power (2.6 to 2.75 W). However, the FFT accelerator relies heavily on the CPU, therefore the power increase of a larger FFT is limited.

Figure 11 shows the energy and power measurements for the B512-1 and a 64K-point FFT. We limit our measurements to this specific configuration for the sake of visibility. We confirmed that the conclusions achieved for this configuration are representative of other designs. Since variations on power are marginal (max 250 mW), energy savings are huge for FFT designs after 1K. In particular, energy is reduced by 37% for 2K, 45% for 4K and 60% for 8K.

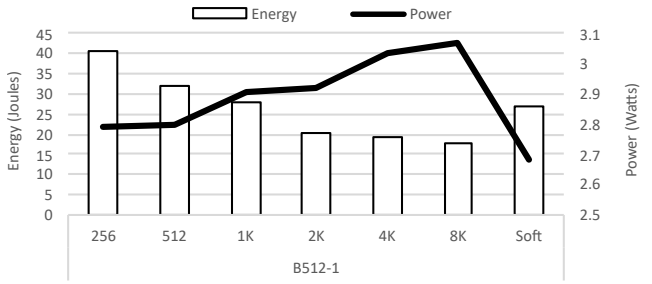

Figure 11: Power and Energy for 64K samples FFT on Zynq 7020

7. Real-World Case Studies

7.1. Oil Well Inspection

Now we describe how our RCS methodology and ScaleFFT accelerator can be leveraged to optimize an oil well inspection system. The purpose of this system is to examine an oil well to identify the quality of its pipes. This is carried out by physically moving the inspection system through the complete well as illustrated in Figure 12. An oil well is typically 1.5 to 3 km long, and the performance of the inspection system is critical as production must be halted during inspection. In other words, the slower the system, the longer it takes to inspect the well, and the longer the facility is idle. The average cost of running an off-shore oil platform varies widely (e.g., from 200k USD per day in 2019 to 250k USD per day in 2020 [57]) but is generally high and minimizing down-time can hence lead to substantial cost savings.

The oil well inspection system faces a rich set of constraints, and this makes it a good example of a resource-constrained edge system. The system needs to perform 288 FFT transforms at least every 20 m to detect anomalies which results in a data rate of 4.2 megabytes per meter (MB/m); one measurement per 20 m is a typical minimum requirement. Upstream bandwidth is limited to 200 Kb/s, which means that inspection speed would be limited to 172 s per meter if the processing was not performed at the edge. For a well length of 2.4 km, which yields 120 measurement points, a complete inspection would hence take 6.7 hours if we assume that it takes one hour to move the system through the well. More specifically, we stop the system for 172 s at each of the 120 measurement points which means that the system spends 5.7 hours collecting samples (and hence 6.7 hours in total when including trip-time). Performing anomaly detection at the edge is hence attractive as it makes the system compute-bound rather than communication-bound, and the compute bandwidth of edge systems is much higher than the upstream bandwidth, i.e., we achieve a processing bandwidth of 1,500 kb/s with the RCS-selected system configuration whereas upstream bandwidth is only 200 kb/s. In our system, anomaly detection is performed using a convolutional neural network.

Our objective was hence to maximize compute performance while respecting all constraints. The first key constraint is the physical size of the system as it needs to fit within the 0.1 m diameter of the well. Second, power consumption is limited to 3.5 W because there is no space for ventilation in the electronics housing and temperatures within an oil well temperature range from 80 C to 170 C. Moreover, this application is important yet low volume as there are a limited number of wells in the world and they only need to be inspected infrequently. For this reason, we selected the Zynq 7000 which contains two hard ARM Cortex-A9 CPU cores and an FPGA fabric as this means that we can accelerate the FFT and neural network inference on a single chip; we could not include a DSP in our system without exceeding the physical size constraint.

We then used our RCS methodology to identify the near-optimal configuration of an 256K-point FFT accelerator and a B512 DPU which yields an end-to-end latency of 22.2 s (see Figure 8). In contrast, the latency of the dual-DPU baseline configuration is 32.6 s. This yields overall inspection times of 1.7 h and 2.1 h for the RCS-optimized configuration and the baseline, respectively, for an oil well depth of 2.4 km (120 points of inspection) and 1 hour of travel time. Our scheme hence reduces downtime by 0.4 h compared to the single-accelerator configuration and 5.0 h compared to the offline processing approach, yielding substantial cost savings of 5.0k and 62.5k USD, respectively, at an average facility (12.5K per hour).

7.2. Train Track Inspection

Railways are a valuable and important infrastructure and it is hence crucial that they have high reliability, i.e., the socio-economic consequences of unexpected downtime can be significant. An attractive way of monitoring the condition of the railway tracks is to install monitoring equipment on the trains that use the track. More specifically, we consider a predictive maintenance system that is attached to the bogies of the train and continuously monitors the tracks. To sense the condition of the track, the system requires high-bandwidth (i.e., 15 to 30 kHz) accelerometers. Moreover, we require Analog to Digital Converters (ADCs) that can capture accelerometer output at 24 bit resolution at sampling rates of 64 to 128 KSPS. Practical systems require multiple accelerometers, and the amount of data captured per sample is hence from 3 to 6 MB. The resulting data rates are typically in the range of 3 to 6 MB/s which means that offline processing is inconvenient. Moreover, the system is installed closed to the bogies and the train is moving at high speed (sometimes through rural or unpopulated areas with limited communication coverage). It is hence necessary to perform the computation at the edge.

We evaluate RCS in the context of a train track inspection system called TrainDAQ [24] which is built around a Trenz Electronics System on Module (SoM) [58] with a Xilinx Zynq 7020 SoC (as shown in Figure 13). TrainDAQ uses an FPGA-based platform because it enables implementing a highly parallel system in the FPGA that captures the output of the ADCs and write it into DDR3 memory using the Zynq's DMA controller. This is critical to achieve the required data rates while maintaining the other constraints, i.e., energy and physical size. While the train has abundant energy available, attaching the monitoring system to the train's power would result in the train having to be re-certified and would hence incur a significant cost overhead. For this reason, a practical track monitoring system must be powered by batteries. For each sample, the system

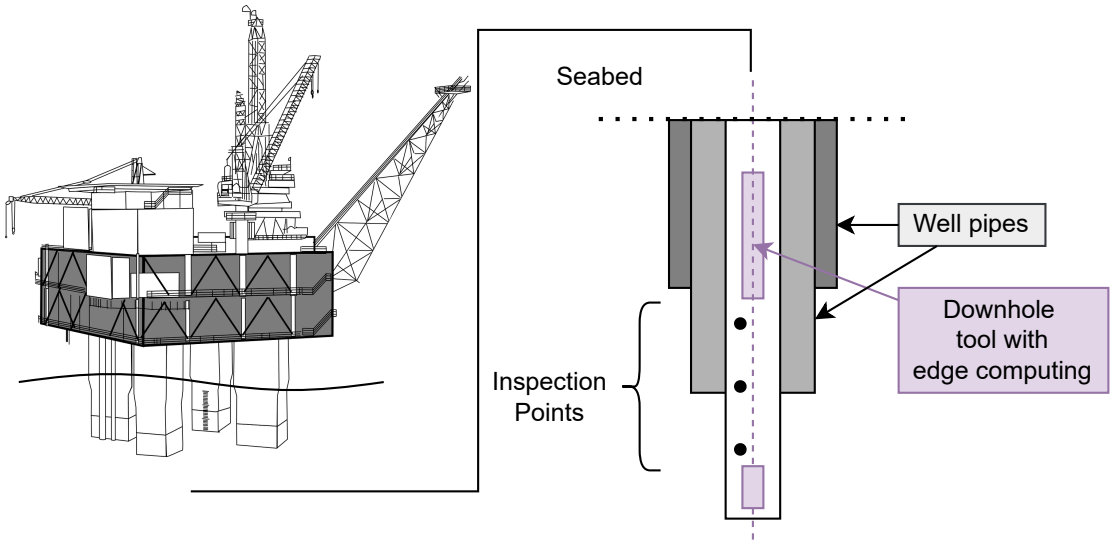

Figure 12: Oil well monitoring case study. *Production must be halted during inspection, and higher performance hence means that the inspection system can be moved faster through the well which in turn reduces inspection time and hence costs.*

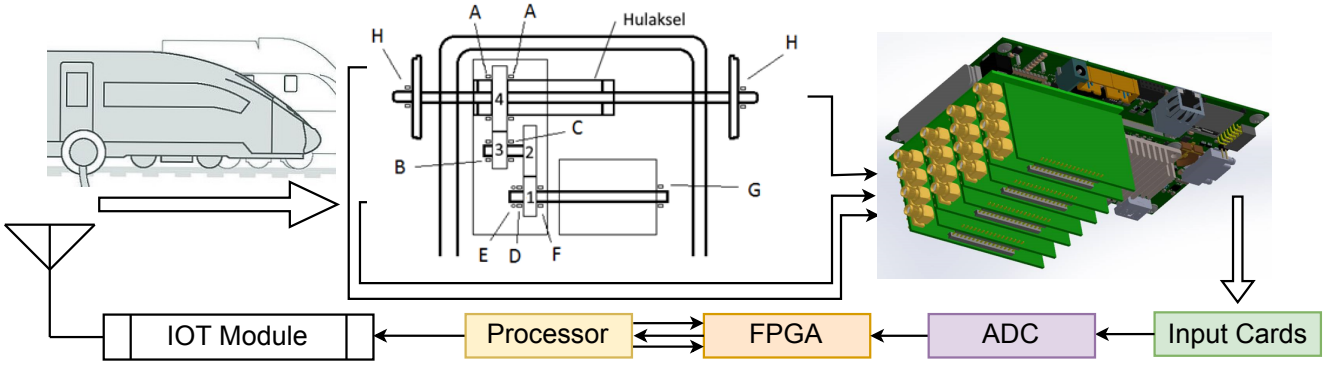

Figure 13: Train Monitoring System

then performs an FFT transform for the output of each sensor and feeds the output of these transforms to the DPU. The output of the DPU will then be a classification of the state of the part of the track that was sampled. This is typically on the order of 4 to 12 KB per sample, depending on the state of the track, which is then transmitted, via the IoT module in Figure 13, to a back-end system if possible. Otherwise, the samples are buffered to be transmitted when connectivity is restored. The IoT module also contains a GPS which is used to map samples to the track segment they cover.

The TrainDAQ deployment that we consider in this work has a 64K point FFT and a B512 DPU. Our RCS methodology selects the dual-DPU and 4K FFT configuration (see Figure 7). The sensors are active for 0.25 s and TrainDAQ then uses 6.5 s to process the sample in the RCS-selected configuration. This results in a spatial resolution of one measurement per 469 meters when the train velocity is 250 km/h. In contrast, the single-DPU baseline uses 10.1 s to process a sample which yields one sample every 722 m; the sensors are active for the same time in both configurations. At 100 km/h, the RCS-selected configuration and the baseline yields spatial resolutions of one sample per 188 m and 288 m, respectively. RCS hence enables us to improve spatial resolution by 54% by simply utilizing the FPGA-resources better.

8. Related Work

For any accelerator, the unique challenge is that it needs to exploit the specifics of the key computation of the target application domain to achieve high efficiency. For an FFT accelerator, the key challenges are hence to efficiently implement the Discrete Fourier Transform (DFT) and the butterfly data exchange (which is what ScaleFFT focuses on).

Signal-processing acceleration is a well studied area in both continuous signal processing on chip multi-processors [59] and within the embedded domain [60]. Accelerators are also proposed for domain-specific scenarios, such as robot motion planning [61], speech recognition [62], and slow sensors [63].

Authors in [64, 65] try to answer how to optimize a computational problem within a given chip's transistor budget. They also provide a model which projects forward to see what future gains can and cannot be enabled from chip specialization. We

focus on FPGA and IP-library optimization with resource limitations, not developing new optimized functions. Many prior accelerator architectures can be approximated by composing a small number of hardware primitives. Accelerators can be generated by searching within such a rich accelerator space, i.e., compiler-based Design Space Exploration (DSE) [66]. They follow a similar DSE procedure to us, but at a different level, i.e., they do not consider resource optimization.

LogCA [11] and NeuroMeter [12] are high-level performance and energy model tools for hardware accelerators, useful to identify design bottlenecks. Similarly, we have our own model extracted from area and performance information extracted from DSE for the elements of the IP-library. Neural network design space exploration for multi-chip designs and workload orchestration can be considered similar to our proposal. DSE commonly chooses an specific allocation across several benchmarks [13]. Analog/mixed-signal machine learning accelerators could be an alternative to FFT transformations (e.g., [67]). Nevertheless they rely on ASIC, so it is not suitable for low volume applications. It could be interesting for our future work to use a similar methodology for FPGAs.

Other related work partitions the available FPGA resources into multiple processors, each tailored for a different subset of the CNN convolutional layers [14, 15, 16]. They also invest significant effort to orchestrate data movement and design efficient mapping strategies. Others offer support for high-level programming, resource estimation, and rapid and automatic design space exploration on FPGAs [68]. We provide a model to estimate end-to-end latency in resource-constrained PdM systems, whereas these works do not discuss what happens when the FPGA does not have enough resources for all processors.

Regarding the need for FFT transformations, authors from [69] show the need to improve memory management for neural networks on edge systems (e.g., move to frequency domain via FFT). Data movement is a key issue when performing inference in edge devices. Authors in [70] proposed an FFT-based deep and distributed memory hierarchy, thus enabling data movement over short wires. Other works propose scheduling DNN accelerators for data reuse [71, 72]. This could be applied to FPGAs, e.g., moving data from FFT to DNN. This however puts further pressure on resource optimization since FPGA resources are also used for storage.

Closest related works that combine FPGA resources for FFT and deep learning include: the use FPGA implementations of an efficient FFT processor for FMCW radar signal processing [17]; a 1 million-point FFT on a single FPGA [18]; an ultra-long FFT architecture implemented in a re-configurable application specified processor [19]; and accelerating convolutional neural network with FFT on embedded hardware [20]. These works are however very domain specific, focusing on either pure hardware ASIC, CUDA, near data computations (e.g., processing in memory) or hardware/software co-design. Our solution is valid for scenarios when volume is low, and it is not feasible to produce ASICs or processing-capable memories. Our design combines hardware and software on FPGA, and differently from all previous works, we offer an optimized design for resource-limited scenarios. In addition, this particular feature makes our design a good candidate for an inline accelerator integrated in the core [73].

9. Conclusion and Future Directions

We have now presented the Resource-Constrained Accelerator Selection (RCS) methodology and the ScaleFFT accelerator architecture which collectively improve the performance of FPGA-accelerated Predictive Maintenance (PdM) systems by allocating FPGA resources to the FFT and neural network accelerators such that end-to-end latency is minimized. RCS effectively identifies near-optimal configurations and reduces end-to-end latency by 2.4× on average for a 256K-point FFT compared to our state-of-the-art baseline which accelerates machine learning inference but runs the FFT in software. We demonstrate the real-world impact of RCS by considering two use cases, i.e., an oil well inspection system and a train track monitoring system. RCS reduces the cost of downtime due to inspection by 5.0k USD for an oil well with 120 inspection points compared to the baseline and improves spatial resolution by 54% in the train track monitoring system.

The accelerator selection problem is not limited to resource-constrained PdM systems. For instance, leading mobile SoCs contain many accelerator types [74], and recent high-performance processor SoCs, such as the Apple M2 [75], contain multiple accelerators. Since RCS exploits the unique characteristics of resource-constrained PdM systems — to provide high efficiency while remaining (relatively) simple — more research is likely needed to identify efficient accelerator configurations in other domains. In particular, we expect that such approaches likely require (i) more advanced accelerator search strategies, and (ii) more comprehensive performance models. We aim to target these challenges in the future.

Acknowledgments

This work has been supported by the Research Council of Norway through the BAMPAM (grant no. 286596) and RailCBM (grant no. 296248) projects.

References

[1] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, T. Zhou, A Survey on Edge Computing Systems and Tools, Proceedings of the IEEE 107 (8) (2019) 1537–1562. doi:10.1109/JPROC.2019.2920341.

[2] G. Gobieski, B. Lucia, N. Beckmann, Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems, in: Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2019, pp. 199–213. doi:10.48550/ARXIV.1810.07751.

[3] D. Durocher, G. Feldmeier, Predictive Versus Preventive Maintenance, IEEE Industry Applications Magazine 10 (5) (2004) 12–21. doi:10.1109/MIA.2004.1330766.

[4] M. Khazraee, L. Zhang, L. Vega, M. B. Taylor, Moonwalk: NRE Optimization in ASIC Clouds, in: Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2017, pp. 511–526. doi:10.1145/3093337.3037749.

[5] Xilinx, Zynq-7000 SoC, https://www.xilinx.com/products/silicon-devices/soc/zynq-7000 (accessed on 22 April 2022).

[6] Intel, Agilex FPGA and SoC FPGA, https://www.intel.com/content/www/us/en/products/details/fpga/agilex.html (accessed on 22 April 2022).

[7] NVIDIA, NVIDIA Tegra: Next Generation Mobile Development, https://developer.nvidia.com/tegra-development (accessed on 22 April 2022).

[8] Texas Instruments, KeyStone Architecture, https://training.ti.com/node/1138812 (accessed on 22 April 2022).

[9] Intel, Intel NUC - Small Form Factor Mini PC, https://www.intel.com/content/www/us/en/products/details/nuc.html (accessed on 22 April 2022).

[10] D. F. Bacon, R. Rabbah, S. Shukla, FPGA Programming for the Masses, Communications of the ACM 56 (4) (2013) 56–63. doi:10.1145/2436256.2436271.

[11] M. S. B. Altaf, D. A. Wood, LogCA: A High-Level Performance Model for Hardware Accelerators, in: Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), 2017, p. 375–388. doi:10.1145/3079856.3080216.

[12] T. Tang, S. Li, L. Nai, N. Jouppi, Y. Xie, NeuroMeter: An Integrated Power, Area, and Timing Modeling Framework for Machine Learning Accelerators Industry Track Paper, in: 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021, pp. 841–853. doi:10.1109/HPCA51647.2021.00075.

[13] Z. Tan, H. Cai, R. Dong, K. Ma, NN-Baton: DNN Workload Orchestration and Chiplet Granularity Exploration for Multichip Accelerators, in: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), 2021, pp. 1013–1026. doi:10.1109/ISCA52012.2021.00083.

[14] Y. Shen, M. Ferdman, P. Milder, Maximizing CNN Accelerator Efficiency Through Resource Partitioning, in: Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), 2017, p. 535–547. doi:10.1145/3079856.3080221.

[15] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, V. Chandra, Heterogeneous Dataflow Accelerators for Multi-DNN Workloads, in: 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021, pp. 71–83. doi:10.1109/HPCA51647.2021.00016.

[16] L. Song, J. Mao, Y. Zhuo, X. Qian, H. Li, Y. Chen, HyPar: Towards Hybrid Parallelism for Deep Learning Accelerator Array, in: 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2019, pp. 56–68. doi:10.1109/HPCA.2019.00027.

[17] J. Heo, Y. Jung, S. Lee, Y. Jung, FPGA Implementation of an Efficient FFT Processor for FMCW Radar Signal Processing, Sensors 21 (19). doi:10.3390/s21196443.

[18] H. Kanders, T. Mellqvist, M. Garrido, K. Palmkvist, O. Gustafsson, A 1 Million-Point FFT on a Single FPGA, Poznan University of Technology Academic Journals. Electrical Engineering 66-I (10) (2019) 3863–3873. doi:10.1109/TCSI.2019.2918403.

[19] F. Han, L. Li, K. Wang, F. Feng, H. Pan, B. Zhang, G. He, J. Lin, An Ultra-Long FFT Architecture Implemented in a Reconfigurable Application Specified Processor, IEICE Electronics Express 13 (13) (2016) 20160504–20160504. doi:10.1587/elex.13.20160504.

[20] T. Abtahi, C. Shea, A. Kulkarni, T. Mohsenin, Accelerating Convolutional Neural Network With FFT on Embedded Hardware, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 26 (9) (2018) 1737–1749. doi:10.1109/TVLSI.2018.2825145.

[21] Xilinx, DPU for Convolutional Neural Network., https://www.xilinx.com/products/intellectual-property/dpu.html#overview (accessed on 22 April 2022).

[22] Xilinx, Fast Fourier Transform v9.1 – LogiCORE IP Product Guide, https://www.xilinx.com/content/dam/xilinx/support/documents/ip_documentation/xfft/v9_1/pg109-xfft.pdf (accessed on 10 October 2022).

[23] Nermina Kulovic, Transocean Rigs Land New Jobs with Better Day Rates, https://www.offshore-energy.biz/transocean-rigs-land-new-jobs-with-better-day-rates/ (accessed on 22 April 2022).

[24] Mantena, Mantena and The Research Council of Norway with exciting collaboration, https://mantena.org/news_en/2021/mantena-and-the-research-council-of-norway-with-exciting-collaboration/ (accessed on 22 April 2022).

[25] A. Garg, V. Vijayaraghavan, K. Tai, P. M. Singru, V. Jain, N. Krishnakumar, Model Development Based on Evolutionary Framework for Condition Monitoring of a Lathe Machine, Measurement 73 (2015) 95–110. doi:10.1016/j.measurement.2015.04.025.

[26] A. Kanawaday, A. Sane, Machine Learning for Predictive Maintenance of Industrial Machines Using IoT Sensor Data, in: 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), 2017, pp. 87–90. doi:10.1109/ICSESS.2017.8342870.

[27] B. Luo, H. Wang, H. Liu, B. Li, F. Peng, Early Fault Detection of Machine Tools Based on Deep Learning and Dynamic Identification, IEEE Transactions on Industrial Electronics 66 (1) (2019) 509–518. doi:10.1109/TIE.2018.2807414.

[28] M. Calabrese, M. Cimmino, F. Fiume, M. Manfrin, L. Romeo, S. Ceccacci, M. Paolanti, G. Toscano, G. Ciandrini, A. Carrotta, M. Mengoni, E. Frontoni, D. Kapetis, SOPHIA: An Event-Based IoT and Machine Learning Architecture for Predictive Maintenance in Industry 4.0, Information 11 (4). doi:10.3390/info11040202.

[29] R. Prytz, S. Nowaczyk, T. Rögnvaldsson, S. Byttner, Predicting the Need for Vehicle Compressor Repairs Using Maintenance Records and Logged Vehicle Data, Engineering Applications of Artificial Intelligence 41 (2015) 139–150. doi:10.1016/j.engappai.2015.02.009.

[30] S. Hong, Z. Zhou, Application of Gaussian Process Regression for Bearing Degradation Assessment, 2012 6th International Conference on New Trends in Information Science, Service Science and Data Mining (ISSDM2012) (2012) 644–648.

[31] G. K. Durbhaka, B. Selvaraj, Predictive Maintenance for Wind Turbine Diagnostics Using Vibration Signal Analysis Based on Collaborative Recommendation Approach, in: 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2016, pp. 1839–1842. doi:10.1109/ICACCI.2016.7732316.

[32] N. Kolokas, T. Vafeiadis, D. Ioannidis, D. Tzovaras, Forecasting Faults of Industrial Equipment Using Machine Learning Classifiers, in: 2018 Innovations in Intelligent Systems and Applications (INISTA), 2018, pp. 1–6. doi:10.1109/INISTA.2018.8466309.

[33] F. De Vita, D. Bruneo, S. K. Das, A Novel Data Collection Framework for Telemetry and Anomaly Detection in Industrial IoT Systems, in: 2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI), 2020, pp. 245–251. doi:10.1109/IoTDI49375.2020.00032.

[34] K. S. Kiangala, Z. Wang, Initiating Predictive Maintenance for a Conveyor Motor in a Bottling Plant Using industry 4.0 Concepts, The International Journal of Advanced Manufacturing Technology 97 (2018) 3251–3271. doi:10.1007/s00170-018-2093-8.

[35] T. dos Santos, F. J. T. E. Ferreira, J. M. Pires, C. Damásio, Stator Winding Short-Circuit Fault Diagnosis in Induction Motors Using Random Forest, in: 2017 IEEE International Electric Machines and Drives Conference (IEMDC), 2017, pp. 1–8. doi:10.1109/IEMDC.2017.8002350.

[36] S. Biswal, G. Sabareesh, Design and Development of a Wind Turbine Test Rig for Condition Monitoring Studies, in: 2015 International Conference on Industrial Instrumentation and Control (ICIC), 2015, pp. 891–896. doi:10.1109/IIC.2015.7150869.

[37] O. Aydin, S. Guldamlasioglu, Using LSTM Networks to Predict Engine Condition on Large Scale Data Processing Framework, in: 2017 4th International Conference on Electrical and Electronic Engineering (ICEEE), 2017, pp. 281–285. doi:10.1109/ICEEE2.2017.7935834.

[38] R. G. Vasconcelos Machado, H. de Oliveira Mota, Simple Self-Scalable Grid Classifier for Signal Denoising in Digital Processing Systems, in: 2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP), 2015, pp. 1057–1061. doi:10.1109/GlobalSIP.2015.7418359.

[39] S. Eke, T. Aka-Ngnui, G. Clerc, I. Fofana, Characterization of the Operating Periods of a Power Transformer by Clustering the Dissolved Gas Data, in: 2017 IEEE 11th International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives (SDEMPED), 2017, pp. 298–303. doi:10.1109/DEMPED.2017.8062371.

[40] T. Huuhtanen, A. Jung, Predictive Maintenance of Photovoltaic Panels via Deep Learning, in: 2018 IEEE Data Science Workshop (DSW), 2018, pp. 66–70. doi:10.1109/DSW.2018.8439898.

[41] M. W. Hoffmann, S. Wildermuth, R. Gitzel, A. Boyaci, J. Gebhardt, H. Kaul, I. Amihai, B. Forg, M. Suriyah, T. Leibfried, V. Stich, J. Hicking, M. Bremer, L. Kaminski, D. Beverungen, P. zur Heiden, T. Tornede, In-

tegration of Novel Sensors and Machine Learning for Predictive Maintenance in Medium Voltage Switchgear to Enable the Energy and Mobility Revolutions, Sensors 20 (7). doi:10.3390/s20072099.

[42] V. Mathew, T. Toby, V. Singh, B. M. Rao, M. G. Kumar, Prediction of Remaining Useful Lifetime (RUL) of Turbofan Engine Using Machine Learning, in: 2017 IEEE International Conference on Circuits and Systems (ICCS), 2017, pp. 306–311. doi:10.1109/ICCS1.2017.8326010.

[43] C. Zhou, C.-K. Tham, GraphEL: A Graph-Based Ensemble Learning Method for Distributed Diagnostics and Prognostics in the Industrial Internet of Things, in: 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), 2018, pp. 903–909. doi:10.1109/PADSW.2018.8644943.

[44] A. Rivas, J. M. Fraile, P. Chamoso, A. González-Briones, I. Sittón, J. M. Corchado, A Predictive Maintenance Model Using Recurrent Neural Networks, in: SOCO, 2019. doi:10.1007/978-3-030-20055-8_25.

[45] P. Adhikari, H. G. Rao, M. Buderath, Machine Learning based Data Driven Diagnostics & Prognostics Framework for Aircraft Predictive Maintenance, in: 10th International Symposium on NDT in Aerospace Dresden, Germany, 2018. doi:10.1109/icton51198.2020.9203551.

[46] H. Li, D. Parikh, Q. He, B. Qian, Z. Li, D. Fang, A. Hampapur, Improving Rail Network Velocity: A Machine Learning Approach to Predictive Maintenance, Transportation Research Part C: Emerging Technologies 45 (2014) 17–26. doi:10.1016/j.trc.2014.04.013.

[47] A. Lasisi, N. Attoh-Okine, Principal Components Analysis and Track Quality Index: A Machine Learning Approach, Transportation Research Part C: Emerging Technologies 91 (2018) 230–248. doi:10.1016/j.trc.2018.04.001.

[48] I. Daniyan, K. Mpofu, M. Oyesola, B. Ramatsetse, A. Adeodu, Artificial Intelligence for Predictive Maintenance in the Railcar Learning Factories, Procedia Manufacturing 45 (2020) 13–18, learning Factories across the value chain – from innovation to service – The 10th Conference on Learning Factories 2020. doi:10.1016/j.promfg.2020.04.032.

[49] R. Onanena, F. Chamroukhi, L. Oukhellou, D. Candusso, P. Aknin, D. Hissel, Estimation of Fuel Cell Life Time Using Latent Variables in Regression Context, in: 2009 International Conference on Machine Learning and Applications, 2009, pp. 632–637. doi:10.1109/ICMLA.2009.35.

[50] T. Praveenkumar, M. Saimurugan, P. Krishnakumar, K. Ramachandran, Fault Diagnosis of Automobile Gearbox Based on Machine Learning Techniques, Procedia Engineering 97 (2014) 2092–2098, "12th Global Congress on Manufacturing and Management" GCMM - 2014. doi:10.1016/j.proeng.2014.12.452.

[51] T. P. Carvalho, F. A. A. M. N. Soares, R. Vita, R. da P. Francisco, J. P. Basto, S. G. S. Alcalá, A Systematic Literature Review of Machine Learning Methods Applied to Predictive Maintenance, Computers and Industrial Engineering 137 (2019) 106024. doi:10.1016/j.cie.2019.106024.

[52] J. Dalzochio, R. Kunst, E. Pignaton, A. Binotto, S. Sanyal, J. Favilla, J. Barbosa, Machine Learning and Reasoning for Predictive Maintenance in Industry 4.0: Current Status and Challenges, Computers in Industry 123 (2020) 103298. doi:10.1016/j.compind.2020.103298.

[53] G. M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, in: Proceedings of the Spring Joint Computer Conference (AFIPS), 1967, pp. 483–485.

[54] M. Koraei, O. Fatemi, M. Jahre, DCMI: A scalable strategy for accelerating iterative stencil loops on FPGAs, ACM Transactions on Architecture and Code Optimization 16 (4) (2019) 1–24.

[55] J. W. Cooley, J. W. Tukey, An Algorithm for the Machine Computation of the Complex Fourier Series, Mathematics of Computation 19 (1965) 297–301.

[56] M. Garrido, A Survey on Pipelined FFT Hardware Architectures, Journal of Signal Processing Systemsdoi:10.1007/s11265-021-01655-1.

[57] Energy Today, S&P Ratings: Offshore Rig Rates to Remain Low Till 2021, https://www.offshore-energy.biz/sp-ratings-offshore-rig-rates-to-remain-low-till-2021/ (accessed on 22 April 2022).

[58] Trenz Electronic, Trenz Electronic TE0701 Carrier Board, https://wiki.trenz-electronic.de/display/PD/TE0701+TRM (accessed on 22 April 2022).

[59] B. Belhadj, A. Joubert, Z. Li, R. Héliot, O. Temam, Continuous Real-World Inputs Can Open Up Alternative Accelerator Designs, in: Proceedings of the International Symposium on Computer Architecture (ISCA), 2013, pp. 1–12. doi:10.1145/2485922.2485923.

[60] D. Liaqat, S. Jingoi, E. de Lara, A. Goel, W. To, K. Lee, I. De Moraes Garcia, M. Saldana, Sidewinder: An Energy Efficient and Developer Friendly Heterogeneous Architecture for Continuous Mobile Sensing, in: Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2016, pp. 205–215. doi:10.1145/2980024.2872398.

[61] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, D. J. Sorin, The Microarchitecture of a Real-Time Robot Motion Planning Accelerator, in: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016, pp. 1–12. doi:10.1109/MICRO.2016.7783748.

[62] R. Yazdani, A. Segura, J. M. Arnau, A. Gonzalez, An Ultra Low-Power Hardware Accelerator for Automatic Speech Recognition, in: Proceedings of the International Symposium on Microarchitecture (MICRO), 2016, pp. 1–12. doi:10.1109/MICRO.2016.7783750.

[63] A. Wang, L. Chen, W. Xu, XPro: A Cross-End Processing Architecture for Data Analytics in Wearables, in: Proceedings of the International Symposium on Computer Architecture (ISCA), 2017, pp. 69–80. doi:10.1145/3140659.3080219.

[64] A. Fuchs, D. Wentzlaff, The Accelerator Wall: Limits of Chip Specialization, in: 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2019, pp. 1–14. doi:10.1109/HPCA.2019.00023.

[65] S. Kumar, N. Sumner, V. Srinivasan, S. Margerm, A. Shriraman, Needle: Leveraging Program Analysis to Analyze and Extract Accelerators from Whole Programs, in: 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017, pp. 565–576. doi:10.1109/HPCA.2017.59.

[66] J. Weng, S. Liu, V. Dadu, Z. Wang, P. Shah, T. Nowatzki, DSAGEN: Synthesizing Programmable Spatial Accelerators, in: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), 2020, pp. 268–281. doi:10.1109/ISCA45697.2020.00032.

[67] P. Srivastava, M. Kang, S. K. Gonugondla, S. Lim, J. Choi, V. Adve, N. S. Kim, N. Shanbhag, PROMISE: An End-to-End Design of a Programmable Mixed-Signal Accelerator for Machine-Learning Algorithms, in: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), 2018, pp. 43–56. doi:10.1109/ISCA.2018.00015.

[68] D. Koeplinger, R. Prabhakar, Y. Zhang, C. Delimitrou, C. Kozyrakis, K. Olukotun, Automatic Generation of Efficient Accelerators for Reconfigurable Hardware, in: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016, pp. 115–127. doi:10.1109/ISCA.2016.20.

[69] M.-Z. Ji, W.-C. Tseng, T.-J. Wu, B.-R. Lin, C.-H. Chen, Micro Darknet For Inference: ESL Reference for Inference Accelerator Design, in: 2019 International SoC Design Conference (ISOCC), 2019, pp. 69–70. doi:10.1109/ISOCC47750.2019.9027644.

[70] S. Gudaparthi, S. Narayanan, R. Balasubramonian, E. Giacomin, H. Kambalasubramanyam, P.-E. Gaillardon, Wire-Aware Architecture and Dataflow for CNN Accelerators, in: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2019, p. 1–13. doi:10.1145/3352460.3358316.

[71] Q. Huang, M. Kang, G. Dinh, T. Norell, A. Kalaiah, J. Demmel, J. Wawrzynek, Y. S. Shao, CoSA: Scheduling by Constrained Optimization for Spatial Accelerators (2021). doi:10.48550/ARXIV.2105.01898.

[72] X. Chen, Y. Han, Y. Wang, Communication Lower Bound in Convolution Accelerators, in: 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020, pp. 529–541. doi:10.1109/HPCA47549.2020.00050.

[73] D. Trilla, J.-D. Wellman, A. Buyuktosunoglu, P. Bose, NOVIA: A Framework for Discovering Non-Conventional Inline Accelerators, in: MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '21, 2021, p. 507–521. doi:10.1145/3466752.3480094.

[74] M. D. Hill, V. J. Reddi, Accelerator-level parallelism, Communications of the ACM 64 (12) (2021) 36–38.

[75] Apple, MacBook Air: M2 chip model tech specs, https://www.apple.com/mt/macbook-air-m2/specs/ (accessed on 11 August 2022).