semiconductors and heterogeneous integration

MDPI

Article

Offloading Boundaries in a Heterogeneous Embedded SoC: Characterization on CPU-FPGA

Anna Sabatini [1,*], Roberto Bagnato [1], Danilo Pietro Pau [2] and Luca Vollero [1]

[1] Research Unit of Artificial Intelligence and Computing Systems, Department of Engineering, Università Campus Bio-Medico di Roma, Via Álvaro del Portillo 21, 00128 Rome, Italy; roberto.bagnato@unicampus.it (R.B.); l.vollero@unicampus.it (L.V.)
[2] System Research and Applications, STMicroelectronics, 20864 Agrate Brianza, Italy; danilo.pau@st.com
* Correspondence: a.sabatini@unicampus.it

Abstract

Systems-on-chip that combine a multicore processor with programmable logic (PL), the FPGA part of the chip, allow computation to be offloaded to an accelerator built in the PL. Whether this pays off depends on the workload and on the cost of moving data. On an AMD Kria KV260, we measure the offloading boundary, i.e., the problem sizes and computational intensities beyond which the accelerator outperforms the processor, and test whether a model predicts it. A synthetic microbenchmark, whose computation per element is set by a parameter K, runs on one and four Arm Cortex-A53 cores and on two accelerators generated by high-level synthesis. A three-parameter model reproduced the accelerator times within 0.04% and predicts that, at 100 MHz, the PL must process more than about five elements in parallel to outperform one core and about 21 to outperform four. A four-lane accelerator confirmed the predicted computation rate but, without burst memory transfers, was 3.3–77 times slower than one core. Under controlled measurement conditions, no accelerator outperformed four cores in time or energy: the better one was 13–22 times slower and used about 10–13 times more energy. Memory-access overheads and the processor baseline are thus first-order design parameters.

Keywords: heterogeneous computing; CPU–FPGA offloading; system-on-chip; Zynq UltraScale+ MPSoC; high-level synthesis; performance modeling; communication overhead; computational intensity; embedded systems; heterogeneous integration

1. Introduction

With the end of Dennard scaling and the slowdown of Moore's law, gains in performance and energy efficiency increasingly come from specialization and from the integration of heterogeneous components rather than from transistor scaling alone [1,2]. Heterogeneous integration, i.e., the combination of logic, memory, accelerators and sensors within a single die or package, is now recognized as a key enabler of post-Moore systems [3–6]. At the system level, however, a heterogeneous platform delivers its potential only if each workload is placed on the component that executes it most efficiently, and this decision must account not only for raw compute throughput but also for the cost of moving data between components [7,8].

Field-programmable SoCs (FPSoCs) such as the AMD Zynq UltraScale+ MPSoC integrate a multicore Arm processing system (PS) and programmable logic (PL) on a single monolithic die, connected through on-chip AXI interfaces to a shared DDR memory [9].

Check for updates

Received: 7 October 2026
Revised: 12 November 2026
Accepted: 17 November 2026
Published: 20 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

They are a tightly coupled and commercially mature instance of heterogeneous integration and, with system-on-module products such as the Kria K26 and its KV260 starter kit [10,11] combined with high-level synthesis (HLS) [12–15], they have become accessible to developers without a hardware-design background. Tight coupling, however, does not make offloading free. An accelerator must be configured through memory-mapped registers, its input and output data must be made visible across the PS–PL boundary either by explicit cache maintenance or through coherent ports, and the PL typically operates at a fraction of the processor clock frequency [16–19].

The offloading decision is therefore a trade-off between the spatial parallelism available in the PL and the fixed and data-proportional costs of communication. Several ingredients of this trade-off have been studied in isolation. Roofline-style models relate attainable performance to operational intensity [20] and have been extended to FPGA designs produced by HLS [21,22]; LogCA models the break-even granularity of generic accelerators from latency, overhead and computational-index parameters [23]; analytical offload models have recently been used to size the number of accelerator clusters of a many-core MPSoC [24]; and a number of experimental studies have characterized processor–FPGA communication on Zynq, Cyclone V and server-class CPU–FPGA platforms [16,19,25–29]. Closest to our goal, Kroh and Diessel proposed a model to predict profitable CPU–FPGA partitioning for short transfers on tightly coupled platforms [30]. To the best of our knowledge, however, existing studies do not jointly (i) sweep data size and computational intensity on a recent tightly coupled MPSoC, (ii) compare the accelerator against both single-core and properly parallelized multicore processor baselines running bit-identical code, and (iii) derive from a small number of calibrated parameters the resulting offloading boundary and the minimum accelerator parallelism required to reach it, expressed in quantities that an HLS designer controls (the number of parallel lanes and the PL clock frequency), and test which parts of this prediction hold for a design that was not used in the calibration. The profitability condition itself is a specialization of established throughput and offload models [20,23]; the contribution of this work lies in its use as a pre-implementation sizing rule and in the experimental evidence on which of its terms can be predicted and which must be measured.

This paper addresses this gap with a parametric microbenchmark that acts as a probe of the platform rather than as an application accelerator. Its computational intensity is controlled by a single parameter K while the transferred data volume per element remains constant, so that the compute-to-communication ratio can be swept independently of the problem size N. The characterization of the CPU–FPGA platform is thus the means, and the offloading boundaries are the result: the measurements locate the boundaries on the grid of configurations, and a model calibrated on the same measurements predicts how they move when the accelerator design changes. The study is organized around three research questions:

- **RQ1—Offloading profitability:** How do problem size and compute-to-communication ratio affect the profitability of offloading to the PL of a tightly coupled CPU–FPGA SoC?
- **RQ2—Communication:** How do fixed invocation costs, cache maintenance and memory-interface width shape the offloading boundary?
- **RQ3—Predictability:** Can a lightweight analytical model, calibrated with a small number of measurements, predict the offloading boundary?

The main contributions of this work are the following:

1. A reproducible, end-to-end measurement setup on the AMD Kria KV260, based on a Linux user-space host, an AXI4-Lite control path and a physically contiguous DMA buffer, in which each of 1260 accelerator runs is verified bit-exactly against a software reference, together with the diagnosis of a silent interface-width mismatch between the processor and the logic that discards control-register writes.

2. A hardware characterization of a baseline HLS accelerator over 30 configurations spanning three orders of magnitude of problem size and two of computational intensity, showing a deterministic cost of $K + 5$ PL cycles per element that is captured by a three-parameter model with a maximum error of 0.04%.
3. An analytical profitability condition that relates PL parallelism, PL clock frequency and the per-iteration cost of the processor, predicting the minimum number of parallel lanes required to outperform single-core and multicore baselines, together with its experimental assessment on a four-lane accelerator, which confirmed the predicted compute scaling exactly but revealed an 82-fold increase in the per-group overhead caused by the loss of burst transfers.
4. A decomposition of the end-to-end offloading cost, a measured offloading map and energy measurements under controlled conditions, showing that the fixed invocation cost is dominated by the software path of the cache-maintenance requests, that no measured accelerator is faster or more energy-efficient than four cores, and that uncontrolled measurement conditions can distort the multicore baseline enough to create spurious offloading regions.
5. A discussion, explicitly separated from the experimentally validated findings, of how the same communication-aware reasoning could apply to other heterogeneous integration scenarios, such as accelerators placed on separate chiplets.

The remainder of the paper is organized as follows. Section 2 reviews background and related work. Section 3 first gives an overview of the methodology and then describes the platform, the benchmark, the accelerator designs, the measurement protocol and the performance model. Section 4 presents the experimental results, which are discussed in Section 5. Section 6 concludes the paper.

2. Background and Related Work

2.1. Tightly Coupled CPU–FPGA SoCs

In the Zynq UltraScale+ MPSoC, the PS contains an application processing unit (APU) with four Arm Cortex-A53 cores sharing an L2 cache, a real-time processing unit and a DDR memory controller, while the PL provides configurable logic, block and UltraRAM memories and DSP slices [9]. The two domains communicate through several families of AXI interfaces. PS-master ports (e.g., `M_AXI_HPM0_FPD`) allow the processor to access registers and memories implemented in the PL. PL-master ports give accelerators direct access to the PS memory system: the high-performance `S_AXI_HP` ports are not coherent with the processor caches, the `S_AXI_HPC` ports are I/O-coherent through the cache-coherent interconnect, and the accelerator coherency port (ACP) provides coherent access through the APU L2 cache [9]. The choice of port determines latency, bandwidth and the amount of software cache maintenance required, and it has been shown to change the total execution time of offloaded kernels significantly [17,18,31]. The Kria K26 system-on-module packages an XCK26 device of this family together with 4 GB of 64-bit DDR4 memory [10], and the KV260 starter kit provides a carrier board and a Linux-based flow for loading accelerators at run time [11].

2.2. Characterization of Processor–FPGA Communication

Choi et al. used microbenchmarks to compare the effective bandwidth and latency of CPU–FPGA platforms with different coupling mechanisms, highlighting that the achievable communication performance depends strongly on the transfer size and on the interface used [25,26]. Weisz et al. studied latency-bound, pointer-chasing access patterns on shared-memory processor–FPGA systems [32]. For FPSoCs, Göbel et al. analysed the memory and communication architecture of Zynq-7000 and Cyclone V SoCs [16], and

Molanes et al. characterized processor–FPGA transfers on Cyclone V and compared low-cost FPSoC families, deriving design guidelines for selecting the transfer mechanism [19,27]. On the Zynq UltraScale+, Min et al. showed that the choice of I/O cache-coherency strategy can change the total execution time by more than a factor of three and proposed a per-data-path selection method [18], Micallef et al. built HLS-compatible stream links between the embedded processor and the PL on top of the ACP [31], and Belloch et al. evaluated the computational performance of the processor, GPU and FPGA components of the device on image-processing and matrix-multiplication benchmarks [28]. More recently, Bolat et al. separated the connection-setup latency of lookaside accelerators on an Arm-based FPGA SoC from the data-transfer time and modelled when offloading outperforms the processor [29], and Berrazueta-Mena and Navas proposed a set of HLS benchmark accelerators for the rapid evaluation of Zynq SoCs [33]. On the KV260 itself, Li and Chen reported that a matrix-multiplication accelerator about seven times faster than software on the Arm cores at kernel level yielded only about a twofold speedup at application level [34], and Costa and Brito showed that the throughput of deep-learning inference on the PL stops scaling when the pre- and post-processing threads saturate the four Cortex-A53 cores [35]. These works quantify the communication substrate in detail but do not, in general, relate it to a workload-dependent offloading decision.

2.3. Performance Models for Accelerators and Offloading

The Roofline model bounds attainable performance by the peak compute rate and by the product of memory bandwidth and operational intensity [20]; da Silva et al. and Siracusa et al. adapted it to HLS-generated FPGA designs to guide optimization [21,22]. Roofline-type models, however, describe a single device and do not include the cost of invoking an accelerator from a host. LogCA explicitly models host–accelerator interaction through latency, overhead, granularity and acceleration parameters, and derives the granularity at which offloading breaks even [23]. Gregg and Hazelwood showed for CPU–GPU systems that speedups reported without data-transfer costs can be misleading [8], while Cota et al. and Shao et al. demonstrated that the way an accelerator is coupled to the SoC memory system can dominate its benefit [36,37]. For CPU–FPGA systems, Huang et al. analysed and modelled collaborative execution strategies [38], and Kroh and Diessel proposed a short-transfer model to predict the optimal partitioning between processor and logic on tightly coupled platforms [30], later exploring fine-grained processor–logic interaction on the cache-coherent Zynq [39]. Colagrande and Benini derived an analytical runtime model of offloaded kernels in a many-core MPSoC, including the dispatch overhead, and used it to determine the minimum number of accelerator clusters needed to meet a runtime constraint [24]; this use of an offload model to size the accelerator is the closest in spirit to the minimum-parallelism condition derived here.

2.4. Reported Offloading Guidelines

Table 1 summarizes the rules and recommendations on offloading that the works above report explicitly. All the contributions agree that fixed and data-proportional communication costs must be part of the offloading decision, and that reporting kernel time alone overstates the benefit of an accelerator. Most of them also express the decision in terms of transfer size or offload granularity (workload). Only Colagrande and Benini use their model to size the accelerator itself, for a many-core MPSoC evaluated in RTL simulation [24]; how much spatial parallelism an accelerator in the PL needs relative to a multicore host, which is the quantity an HLS designer controls directly, is not stated for FPSoCs.

Table 1. Offloading rules and recommendations reported in the literature.

Reference	Platform	Reported Rule or Recommendation
Gregg and Hazelwood [8]	CPU–GPU	Speedups must include data-transfer time and state where the data reside; kernel-only comparisons can be misleading.
Altaf and Wood (LogCA) [23]	Generic accelerators	Offload only above a break-even granularity set by latency, overhead and acceleration; at small granularity, overheads rather than acceleration limit the speedup.
Choi et al. [25,26]	Server CPU–FPGA	Effective bandwidth and latency depend strongly on payload size and coupling mechanism; small transfers are latency-bound.
Sadri et al. [17]; Min et al. [18]	Zynq-7000; Zynq UltraScale+	The choice between coherent and non-coherent ports changes execution time substantially and should be made per data path.
Molanes et al. [19,27]	Cyclone V and low-cost FPSoCs	Design guidelines for selecting the processor–FPGA transfer mechanism according to transfer size.
Cota et al. [36]	SoC accelerators (simulation)	The coupling of the accelerator to the memory hierarchy can dominate its benefit.
Kroh and Diessel [30]	Zynq	Short transfers can be modelled to predict when partitioning between processor and logic pays off.
Bolat et al. [29]	Arm-based FPGA SoC	Connection setup and data transfer are separate costs; offloading pays off only when the saved computation exceeds both.
Colagrande and Benini [24]	Many-core MPSoC (RTL simulation)	Dispatch overhead grows with the number of accelerator clusters; a runtime model gives the minimum number of clusters for a runtime target.
Li and Chen [34]	Kria KV260	Kernel-level speedups shrink at application level once host-side work and transfers are included.

2.5. Positioning of This Work

The present study combines elements of these lines of work in a deliberately minimal setting. It uses a single kernel whose computational intensity is an explicit parameter, executes bit-identical code on one core, on four cores and in the PL of a recent tightly coupled MPSoC, and calibrates a model with few parameters that is then used to predict, before implementation, how much PL parallelism is required to make offloading profitable against each processor baseline. The goal is to provide a transparent methodology that exposes where, and why, the offloading boundary lies.

Comparing this work with the presented one, it is possible to highlight differences related to how the acceleration is parameterized and used.

3. Materials and Methods

3.1. Overview of the Methodology

Figure 1 summarizes the methodology. A single parametric kernel, whose problem size N and computational intensity K can be set independently, is executed on three targets: one processor core (CPU-1), four processor cores (CPU-4) and an accelerator in the PL. For every configuration, each target is timed, and every accelerator output is checked against

the processor result. The measurements calibrate a model with few parameters, from which the minimum accelerator parallelism, the break-even intensity and the minimum problem size for profitable offloading are predicted before a new accelerator is designed. A second accelerator, not used in the first calibration, is then implemented to test the prediction; a term that fails the test is traced back and recalibrated. The same procedure applies to another kernel by replacing its inner loop: the calibrated numbers change; the steps do not. Figure 2 in Section 3.6 shows the corresponding hardware setup.

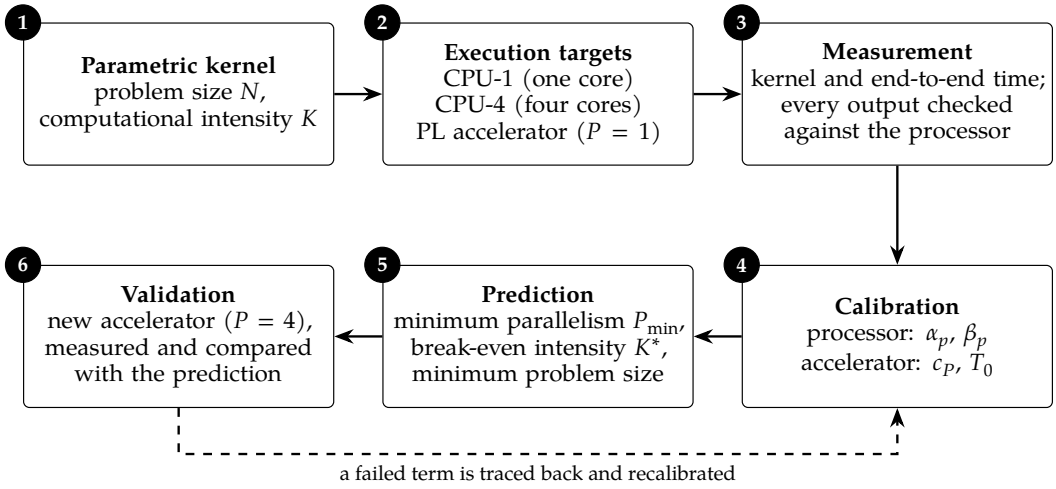


Figure 1. Overview of the methodology. Steps 1–3 produce verified timings for the processor baselines and a first accelerator; step 4 fits the model of Section 3.9; step 5 predicts, before implementation, how much accelerator parallelism and which problem sizes are needed for offloading to pay off; step 6 tests the prediction on a new design.

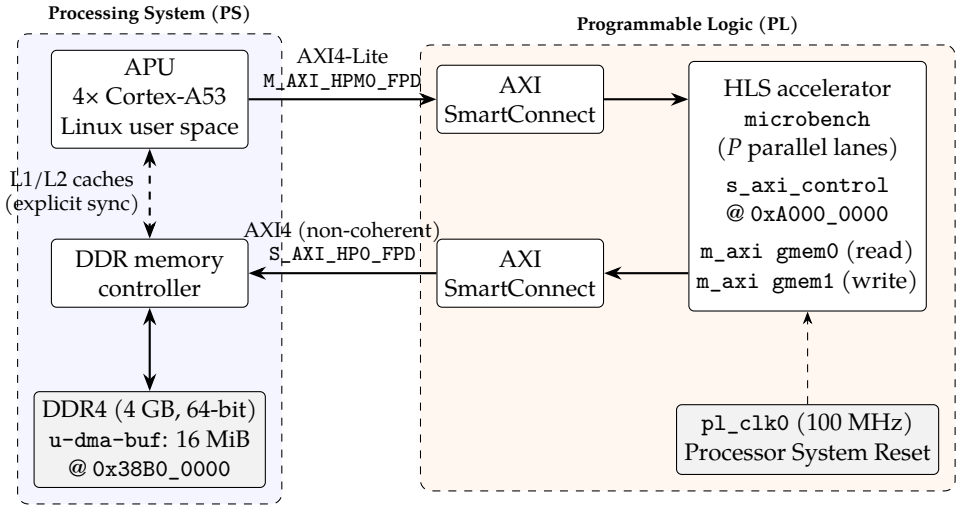


Figure 2. System architecture on the Kria KV260. The processor configures the accelerator through an AXI4-Lite control path, while the accelerator reads and writes a physically contiguous DMA buffer in DDR memory through the non-coherent high-performance port `S_AXI_HP0_FPD`.

3.2. Experimental Platform

All experiments were performed on an AMD Kria KV260 Vision AI Starter Kit [11], whose K26 system-on-module hosts an XCK26-SFVC784-2LV-C Zynq UltraScale+ device [10]. Table 2 summarizes the hardware resources and the software stack. The PL clock was generated by the PS (`pl_clk0`); its frequency, read back from Linux, was 99,999,999 Hz, and 100 MHz is used in the following. The APU frequency was fixed at 1333 MHz by the `userspace` `cpufreq` governor throughout the experiments.

Table 2. Hardware and software configuration of the experimental platform.

Item	Configuration
Board/module	AMD Kria KV260 Vision AI Starter Kit, K26 SOM [10,11]
Device	XCK26-SFVC784-2LV-C (Zynq UltraScale+ MPSoC)
Processor (APU)	4× Arm Cortex-A53 [10], fixed at 1333 MHz (`userspace` governor)
Memory	4 GB DDR4, 64-bit, 2400 Mb/s [10]
PL resources	117,120 LUTs; 234,240 flip-flops; 1248 DSP slices; 144 block RAMs; 64 UltraRAMs [10]
PL clock	`pl_clk0` = 100 MHz (99,999,999 Hz)
Operating system	Ubuntu 22.04, Linux 5.15.0-1027-xilinx-zynqmp
Compiler (host)	GCC/G++ 11.4.0; processor baselines `-O3 -std=c++17` (`-fopenmp` for CPU-4); accelerator host program `-O2`
FPGA tools	AMD Vitis Unified IDE 2025.2 (HLS), Vivado 2025.2
DMA buffer	`u-dma-buf` kernel module [40], 16 MiB, non-coherent, non-cacheable user-space mapping

3.3. Parametric Microbenchmark

The workload is an element-wise transformation of a vector $\mathbf{x} \in \{0, \ldots, 2^{32}-1\}^N$ into a vector $\mathbf{y}$ of the same size. For each element, K iterations of a linear congruential update are accumulated:

$$y_i = \sum_{j=0}^{K-1} \left[(x_i \oplus j) \cdot a + c \right] \bmod 2^{32}, \qquad i = 0, \ldots, N-1, \tag{1}$$

where $\oplus$ denotes the bitwise exclusive OR, all operations are performed on unsigned 32-bit integers, and $a = 1{,}664{,}525$ and $c = 1{,}013{,}904{,}223$ are the classical constants of a 32-bit linear congruential generator [41]. The update is borrowed from a linear congruential generator only because it is a cheap and well-known integer recurrence; the kernel is not used to generate random numbers.

The purpose of the microbenchmark is to probe the platform, not to accelerate an application. In a real kernel, the amount of computation, the data volume, the access pattern and the degree of parallelism change together, so that a measured speedup cannot be attributed to a single cause. Here, one parameter changes the amount of computation per element while everything else stays fixed, which makes it possible to observe directly how the offloading boundary moves with the compute-to-communication ratio and to calibrate a model whose parameters have a clear physical meaning.

The benchmark was designed to have four properties. First, elements are independent, so that the available parallelism grows with N and can be exploited both by threads on the processor and by parallel lanes in the PL. Second, the arithmetic work per element grows linearly with K, whereas the data volume does not: each element requires reading and writing one 32-bit word, i.e., 8 bytes per element regardless of K. With four integer operations per inner iteration (exclusive OR, multiplication, and two additions), the operational intensity is

$$I(K) = \frac{4K \text{ ops}}{8 \text{ bytes}} = \frac{K}{2} \text{ ops/byte}, \tag{2}$$

so that K directly sets the compute-to-communication ratio. Third, integer arithmetic makes processor and accelerator results bit-exact, which allows every hardware run to be verified element by element. Fourth, the dependence of each term on j prevents compilers and HLS tools from replacing the inner loop with a closed-form expression, so that the executed work actually scales with K. The input was initialized as $x_i = i$ in all experiments. The source code of the kernels is reported in Appendix A.

These properties make the microbenchmark favourable to the accelerator in two respects: its parallelism is unlimited, and its memory accesses are perfectly sequential. Where offloading does not pay off for it, it is therefore unlikely to pay off for kernels of similar intensity with less parallelism or irregular accesses. The converse does not hold: kernels with data reuse in on-chip memory, or with arithmetic that maps onto the DSP slices of the PL, may favour the accelerator more than the microbenchmark does (Section 5.7). To apply the methodology to another kernel, the steps of Figure 1 are repeated with its own inner loop: the processor parameters α_p and β_p are measured on the target cores, the accelerator parameters c_P and T_0 are measured on a first, single-lane HLS design, and Equation (8), together with the fixed costs of Equation (3), then gives the minimum parallelism and the minimum problem size for profitable offloading.

3.4. Processor Baselines

Two processor baselines were implemented in C++17 using the same loop nest as the accelerator. *CPU-1* is a single-threaded implementation. *CPU-4* parallelizes the outer loop over elements with an OpenMP `parallel for` directive [42] and runs with four threads (`OMP_NUM_THREADS=4`), one per Cortex-A53 core. Both were compiled with GCC 11.4.0 and `-O3`; as reported by `-fopt-info-vec`, the compiler vectorized the inner loop over j with 128-bit NEON instructions, i.e., four 32-bit lanes. CPU-4 was measured once with unbound threads and in five independent campaigns with threads bound to cores (`OMP_PROC_BIND=close`, `OMP_PLACES=cores`); the CPU-4 value of a configuration is the lowest of these six medians, i.e., the strongest multicore baseline observed, and the spread between the five bound campaigns quantifies its reproducibility. Each measurement consists of one warm-up execution followed by 20 timed repetitions measured with `std::chrono::steady_clock`; the median and the standard deviation are reported. Every output of CPU-1 and CPU-4 was compared element by element with the software reference. The processor baselines were measured on the same grid of problem sizes and intensities as the accelerators (Section 3.8).

3.5. Accelerator Designs

The accelerators were described in C++ and synthesized with AMD Vitis HLS [15] for the XCK26 device with a target clock period of 10 ns. Input and output vectors are accessed through two AXI4 master interfaces (`m_axi`, bundles `gmem0` for reading and `gmem1` for writing), whose base addresses, together with N, K and the block-level control signals, are exposed through an AXI4-Lite slave interface (`s_axilite`). Two variants were developed.

Baseline accelerator ($P = 1$).

The loop nest of Equation (1) is implemented directly. The inner loop over j is pipelined with an initiation interval (II) of one cycle, whereas elements are processed sequentially by the outer loop. The expected cost is therefore of the order of K cycles per element plus a constant per-element overhead due to loop entry and exit and to the memory accesses.

Multi-lane accelerator ($P = 4$).

To exploit the independence of elements, the second variant processes $P = 4$ elements per outer iteration. The operands and accumulators of the four lanes are held in completely

partitioned local arrays, and the inner loop over j is pipelined with II = 1 while the lane body is fully unrolled, so that four independent updates are performed per cycle. Boundary conditions handle values of N that are not multiples of P: unlike in the baseline, the four loads and the four stores of each group are guarded by a comparison of the element index with N (Appendix A). Functional correctness was verified in C simulation and C/RTL co-simulation with $N = 1027$ and $K = 16$, and on the board for all measured configurations (Section 4.5).

In both designs, the two `m_axi` interfaces are 32 bits wide. Table 3 reports the HLS estimates and the post-implementation results. Both designs use no DSP slices, since the multiplication by the constant a is mapped to logic. After implementation, the baseline and four-lane accelerators occupy 1751 and 2817 LUTs and 2607 and 3259 flip-flops, respectively, and both complete PL designs close timing at 100 MHz, with worst negative slacks of +5.49 and +4.96 ns.

Table 3. HLS synthesis estimates and post-implementation results (accelerator only; Vivado 2025.2) for the XCK26 device (target clock period 10 ns). Percentages refer to the resources in Table 2. The slack refers to the complete PL design, including the interconnect.

Metric	$P = 1$	$P = 4$
HLS estimated clock period (ns)	7.3	7.0
Inner-loop II (cycles)	1	1
LUTs (HLS estimate)	2533 (2.2%)	4716 (4.0%)
Flip-flops (HLS estimate)	1905 (0.8%)	2853 (1.2%)
Block RAMs (HLS estimate)	2 BRAM_18K (0.7%)	2 BRAM_18K (0.7%)
DSP slices	0	0
UltraRAMs	0	0
LUTs after implementation	1751 (1.5%)	2817 (2.4%)
Flip-flops after implementation	2607 (1.1%)	3259 (1.4%)
Block RAMs after implementation	1 RAMB18	1 RAMB18
Worst negative slack at 10 ns (ns)	+5.49	+4.96
Burst transfers inferred (HLS log)	yes (`gmem0`, `gmem1`)	none

3.6. System Integration

The accelerator was integrated into a Vivado block design, shown schematically in Figure 2. The control path connects the PS master port `M_AXI_HPM0_FPD` through an AXI SmartConnect to the accelerator's AXI4-Lite interface, which is mapped at base address `0xA000_0000` with a 64 KiB range. The data path connects the two `m_axi` interfaces through a second SmartConnect to the PS slave port `S_AXI_HP0_FPD`, which gives the accelerator access to DDR memory; this port is configured for 128 bits, and the SmartConnect converts from the 32-bit `m_axi` interfaces. `M_AXI_HPM0_FPD` is 32 bits wide in the four-lane design and 128 bits wide in the baseline design (Section 4). The accelerator and the interconnect are clocked by `pl_clk0` and reset by a Processor System Reset block driven by the PS reset output. The `S_AXI_HP0_FPD` port is not coherent with the processor caches, so coherence is maintained in software (Section 3.7). The bitstream was converted with Bootgen and loaded at runtime together with a device-tree overlay through the Kria application-management utilities [11].

3.7. Host Software and Memory Management

A single host program, used for the processor baselines and the accelerators alike, runs in Linux user space. The accelerator registers are accessed by mapping the control address range through `/dev/mem`. Because the accelerator operates on physical addresses, input

and output vectors must reside in physically contiguous memory reachable through the configured HP port. Ordinary user-space allocations do not satisfy this requirement, and an initial attempt based on huge pages returned a physical region outside the low DDR window reachable through the port. The `u-dma-buf` kernel module [40] was therefore used to reserve a physically contiguous buffer in the low DDR region, exposed to user space through `mmap`. All reported measurements used a 16 MiB buffer at physical address `0x38B0_0000`. The buffer is not hardware-coherent. The host opens the device with `O_SYNC`, for which the synchronization mode of the driver used here (`sync_mode` = 1) maps the buffer as non-cacheable in user space [40], and it additionally requests cache maintenance explicitly through the `sync_for_device` and `sync_for_cpu` operations of the driver, so that the sequence is also correct for a cacheable mapping. Each request consists of four writes to `sysfs` attributes of the driver (offset, size, direction and the trigger of the operation), issued on file descriptors that remain open for the whole run. Input and output vectors are placed in adjacent, disjoint regions of the buffer, the output starting at offset $4N$ bytes (e.g., 4 KiB for $N = 1024$ and 4 MiB for $N = 2^{20}$), so that the 16 MiB buffer accommodates problem sizes up to $N = 2^{21}$.

Each accelerator invocation performs the following steps: (1) initialize the input and fill the output region with a poison value (`0xDEADBEEF`); (2) synchronize the buffer for the device; (3) write the physical base addresses, N and K to the control registers and read them back for verification; (4) set `ap_start`; (5) poll the `ap_done` bit; (6) synchronize the buffer for the processor; and (7) compare every output element with the software reference of Equation (1). The host program aborts if any read-back in step (3) differs from the written value.

3.8. Measurement Protocol and Metrics

For the accelerator, the *kernel time* T_{kern} is measured on the host with `std::chrono::steady_clock` from the write of `ap_start` to the observation of `ap_done`. It therefore includes the AXI4-Lite accesses needed to start and poll the accelerator and all DDR traffic generated by the kernel, but excludes buffer preparation and cache maintenance. The *end-to-end time* is decomposed as

$$T_{\text{e2e}} = T_{\text{cfg}} + T_{\text{s2d}} + T_{\text{kern}} + T_{\text{s2c}}, \tag{3}$$

where T_{cfg} is the time to program and read back the six argument registers (step 3), T_{s2d} is the time of the `sync_for_device` request over the input and output regions ($8N$ bytes, step 2) and T_{s2c} is that of the `sync_for_cpu` request over the output region ($4N$ bytes, step 6). The verification step is excluded from all timings.

Accelerators and processor baselines were evaluated on the same grid, $N \in \{2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}\}$ and $K \in \{1, 4, 16, 64, 256\}$, i.e., 30 configurations, in a single campaign. For each accelerator configuration, one warm-up invocation and 20 timed invocations were executed in the same process, and the median of each term of Equation (3) is reported; the processor baselines follow the protocol of Section 3.4. Every median is reported together with the standard deviation of the corresponding runs, in the tables and as error bars in the figures.Results are reported as absolute times, as per-element costs $\tau = T/N$, and as PL cycles per element $C = \tau f_{\text{PL}}$. Speedups are defined as $S = T_{\text{CPU}}/T_{\text{acc}}$, where T_{acc} is the accelerator time (kernel or end-to-end, as stated), so that $S > 1$ indicates profitable offloading. Only runs with zero mismatching elements are reported.

Measurement conditions.

All results reported in this paper come from a single controlled campaign. The APU frequency was fixed at 1333 MHz (Section 3.2); the system ran in multi-user mode without a graphical session, periodic maintenance services (package updates, snap and firmware

daemons) were stopped, and the remaining services were idle, each below 1% of one core in a 5 s sample taken before the runs. The network interface remained active, with an idle SSH session used to start the campaign. No cores were isolated from the operating system; the benchmark ran at the highest user priority (nice −20), and the power and temperature logger described below ran as a real-time task so that it kept sampling when all four cores were busy. The on-chip temperature sensors remained between 23 and 31 °C (PL: 23–26 °C) throughout the campaign, with the fan under the default thermal control. Because the PS side of `M_AXI_HPM0_FPD` is not reconfigured when a design is loaded (Section 4.1), the measurement scripts set it explicitly to the width of each design. A first campaign, run on the same board without these precautions, had shown differences up to a factor of 13.6 between repeated CPU-4 measurements of the same configuration and a fixed cost of about 1.9 ms per invocation of CPU-4 with unbound threads; under controlled conditions, both effects disappeared (Section 4.2).

Energy.

The power of the K26 SOM, which comprises the PS, the PL and the DDR memory but not the carrier board, was sampled at 50 Hz from its INA260 monitor through the Linux `hwmon` interface. For each target and each K at $N = 2^{20}$, the kernel was invoked repeatedly for at least 10 s after one warm-up invocation, and the power samples falling within this loop were averaged; the logger and the host program use the same monotonic clock, so that the loop boundaries are exact. Each configuration was measured in three independent repetitions, and the mean ± standard deviation over the repetitions is reported. The energy per element is the mean power multiplied by the loop duration and divided by the number of processed elements; the dynamic energy subtracts the idle power, measured for 60 s with the same design loaded. For the accelerators, the end-to-end invocation of Equation (3) was repeated with the host either polling `ap_done` continuously or sleeping for 20 µs between polls. The energy–delay product (EDP) is computed per invocation.

3.9. Analytical Performance Model

The processor cost is modelled as linear in N and K,

$$T_{\mathrm{CPU},p}(N,K) \;=\; N\,(\alpha_p + \beta_p K), \qquad p \in \{1,4\}, \tag{4}$$

where α_p captures the per-element cost that does not depend on K (loads, stores and loop overhead) and β_p is the effective cost of one inner iteration when p cores are used. For an accelerator with P lanes that processes groups of elements sequentially, as both designs of this work do, the kernel time is modelled as

$$T_{\mathrm{kern},P}(N,K) \;=\; T_0 + \left\lceil \frac{N}{P} \right\rceil \frac{K + c_P}{f_{\mathrm{PL}}}, \tag{5}$$

where T_0 is a fixed invocation cost and c_P is the number of non-overlapped overhead cycles per group. Independently of the schedule, the memory interfaces impose a lower bound: with w-bit `m_axi` ports clocked at f_{PL} and 4 bytes read and written per element,

$$T_{\mathrm{kern}}(N) \;\geq\; \frac{4N}{(w/8)\,f_{\mathrm{PL}}}, \tag{6}$$

which corresponds to 10 ns per element for the 32-bit ports of both designs at 100 MHz; the 128-bit HP port does not tighten this bound.

Profitability condition.

For large N, fixed costs vanish, and offloading is profitable against baseline p when the per-element cost of the accelerator is lower than that of the processor:

$$\frac{K + c_P}{P f_{\mathrm{PL}}} \;<\; \alpha_p + \beta_p K. \tag{7}$$

Both sides are linear in K. If $\beta_p > 1/(Pf_{\mathrm{PL}})$, Equation (7) holds for all K above a break-even intensity $K_p^* = \big(c_P/(Pf_{\mathrm{PL}}) - \alpha_p\big)/\big(\beta_p - 1/(Pf_{\mathrm{PL}})\big)$, and for every K if $K_p^* < 0$. If instead $\beta_p < 1/(Pf_{\mathrm{PL}})$, the accelerator can win only at low intensity, and only if its per-element overhead is smaller than that of the processor. A necessary condition for the accelerator to remain profitable as the computational intensity grows is therefore

$$P > P_{\mathrm{min},p} = \frac{1}{\beta_p f_{\mathrm{PL}}}, \tag{8}$$

i.e., the aggregate rate of inner iterations in the PL, Pf_{PL}, must exceed the rate $1/\beta_p$ achieved by the processor baseline. At low intensity, the interface bound of Equation (6) must additionally be below $\alpha_p + \beta_p K$. Finally, for finite N, the fixed costs T_0, T_{cfg} and the size-dependent cache-maintenance terms of Equation (3) determine the minimum problem size for which offloading pays off.

4. Results

4.1. Functional Validation of the Platform

Each layer of the system was validated separately before any timing was recorded. Reading the accelerator control register returned the expected idle state (`ap_idle`, value `0x4`); values written to the N and K registers were read back unchanged; the `u-dma-buf` device reported the expected size and physical address; and all arguments written to the accelerator were read back correctly before every invocation. All 1260 accelerator invocations of the timing campaign (two designs, 30 configurations, one warm-up and 20 timed runs each) and the final outputs of all 60 accelerator energy loops were identical to the software reference, with zero mismatching elements; every processor output was likewise identical to the reference.

Integrating the four-lane accelerator exposed a silent failure mode of the PS–PL control path. In a first deployment, values written to the registers at offsets `0x14`, `0x1C` and `0x28` (upper half of the input address, lower half of the output address and N) were read back as zero, whereas those written at `0x10`, `0x20` and `0x30` were stored correctly. The same pattern appeared with plain `devmem` accesses, which excluded the host program. Only offsets that are multiples of 16 bytes worked, which is the signature of a disagreement between the PS and the PL on the data width of `M_AXI_HPM0_FPD`: the bitstream actually loaded had been generated from an earlier block design with a 128-bit port, whereas the PS side was configured at run time for 32 bits (register `AFI_FS` [9]). In this condition, a 32-bit transfer whose address is not a multiple of 16 bytes travels on byte lanes that the two sides interpret differently, so that the accelerator receives no enabled write strobes and the processor samples read-data lanes that carry no data. Temporarily configuring the PS side for 128 bits made the same accesses succeed, and rebuilding the bitstream from the current design restored correct operation with the 32-bit setting. The baseline accelerator, whose block design uses a 128-bit `M_AXI_HPM0_FPD`, was accordingly measured with the PS side set to 128 bits. Because the application-management utilities do not modify this register when an accelerator is loaded, it must be configured explicitly whenever designs built with different port widths are exchanged; the setting affects only the control path, not the data path through `S_AXI_HP0_FPD`. Since the failure produces no error response, the read-back of all arguments in step (3) of the invocation sequence (Section 3.7) was kept in all measurements, and the host program aborts on any mismatch.

4.2. Processor Baselines

Table 4 reports the processor baselines at $N = 2^{20}$; the full grid is used in Section 4.7. For CPU-1, the per-element cost grows from 13.4 ns at $K = 1$ to 502 ns at $K = 256$ and

becomes nearly proportional to K at high intensity (a factor of 3.6 between $K = 64$ and $K = 256$, for four times the work), confirming that the benchmark effectively controls the amount of work.

Under controlled conditions, the four-thread baseline is regular and reproducible. Its speedup over CPU-1 at $N = 2^{20}$ is 3.92–3.98 at every intensity; over the whole grid, the best CPU-4 configuration is within 10% of ideal four-core scaling in 25 of the 30 configurations. The other five are small problems ($N \leq 2^{12}$ and $K \leq 16$), in which a fork–join overhead of a few microseconds is comparable with the work: at $N = 1024$ and $K = 1$, CPU-4 takes 6.0 µs against 13.2 µs for CPU-1. The medians of the five bound campaigns differ by at most 5% for every configuration, and unbound and bound threads perform alike (ratio of the medians 0.99–1.02). Individual runs still fluctuate: at $N = 2^{20}$, the coefficient of variation of the CPU-4 runs ranges from 1% to 34% (median 7%), against a median of 0.3% for CPU-1, because any concurrent activity on a system with exactly four cores delays one thread and, with it, the whole parallel loop; the median, which is robust to such outliers, is used throughout, together with the standard deviation. These results differ markedly from those of the first campaign (Section 3.8), in which repeated CPU-4 measurements differed by up to a factor of 13.6 and unbound threads paid about 1.9 ms per invocation: uncontrolled conditions affected mainly the multicore baseline.

A least-squares fit of Equation (4) to the CPU-1 costs at $N = 2^{20}$ yields $\alpha_1 = 12.4$ ns and $\beta_1 = 1.92$ ns per iteration; the linear model deviates by up to 15% at the lowest intensities, and restricting the fit to $K \geq 16$ gives $\beta_1 = 1.91$ ns. For CPU-4, a fit over $K \geq 16$ gives $\beta_4 = 0.487$ ns, within 2% of ideal four-core scaling ($\beta_1/4 = 0.48$ ns).

Table 4. Processor baselines for $N = 1{,}048{,}576$ elements (median ± standard deviation of the timed runs after one warm-up). CPU-4 was measured once with unbound threads and in 5 independent campaigns with threads bound to cores; the bound column reports the best campaign, and the spread is the ratio between the slowest and the fastest campaign median. The speedup refers to the best CPU-4 configuration.

K	CPU-1 (ms)	CPU-4, Unbound (ms)	CPU-4, Bound (ms)	Spread of Bound Campaigns	Best CPU-4 Speedup
1	14.10 ± 0.08	3.57 ± 0.71	3.58 ± 1.37	1.00	3.94 ± 0.79
4	18.29 ± 0.14	4.64 ± 0.71	4.63 ± 0.71	1.00	3.95 ± 0.60
16	44.22 ± 0.16	11.28 ± 2.18	11.27 ± 0.88	1.00	3.92 ± 0.31
64	147.95 ± 0.19	37.20 ± 4.20	37.20 ± 1.00	1.00	3.98 ± 0.11
256	526.64 ± 0.26	134.38 ± 4.03	134.45 ± 3.92	1.00	3.92 ± 0.12

4.3. Baseline Accelerator

Table 5 lists the kernel times of the baseline accelerator over the full grid, and Figure 3 compares them with the model. The behaviour is remarkably regular. At $N = 2^{20}$, the measured cost is 6.000, 9.000, 21.000, 69.00 and 261.00 PL cycles per element for $K = 1$, 4, 16, 64 and 256, respectively, i.e., exactly $K + 5$ cycles. A least-squares fit of Equation (5) with $P = 1$ and free per-element and per-iteration coefficients returns 5.0 overhead cycles per element, 1.0000 cycle per inner iteration and a fixed cost $T_0 = 1.55$ µs; with these parameters, the maximum relative error over the 30 configurations is 0.04%. Even without the fixed term, the simple expression $T = N(K + 5)/f_{\mathrm{PL}}$ reproduces all measurements within 2.5%, the largest deviation occurring at $N = 1024$ and $K = 1$. Run-to-run variability is negligible: the coefficient of variation of the 20 runs is at most 0.16% and, for $N \geq 2^{16}$, below 0.003%.

Table 5. Kernel time (µs, median ± standard deviation of 20 runs) of the accelerator with $P = 1$ measured on the KV260 at $f_{PL} = 100$ MHz. All runs produced zero mismatching elements. The model $T = T_0 + N(K + c_1)/f_{PL}$ with $T_0 = 1.55$ µs and $c_1 = 5.00$ reproduces every median within 0.04%.

N	$K = 1$	$K = 4$	$K = 16$	$K = 64$	$K = 256$
1024	63.02 ± 0.10	93.70 ± 0.09	216.64 ± 0.10	708.14 ± 0.08	2674.20 ± 0.11
4096	247.29 ± 0.09	370.24 ± 0.13	861.69 ± 0.10	2827.80 ± 0.07	10,692.20 ± 0.07
16,384	984.64 ± 0.09	1476.10 ± 0.11	3442.21 ± 0.10	11,306.56 ± 0.15	42,764.12 ± 0.10
65,536	3933.73 ± 0.11	5899.80 ± 0.10	13,764.18 ± 0.10	45,221.70 ± 0.13	171,051.64 ± 0.11
262,144	15,730.28 ± 0.08	23,594.73 ± 0.10	55,052.15 ± 0.09	180,882.20 ± 0.11	684,202.26 ± 0.10
1,048,576	62,916.61 ± 0.13	94,374.05 ± 0.12	220,204.10 ± 0.11	723,524.05 ± 0.13	2,736,804.14 ± 0.14

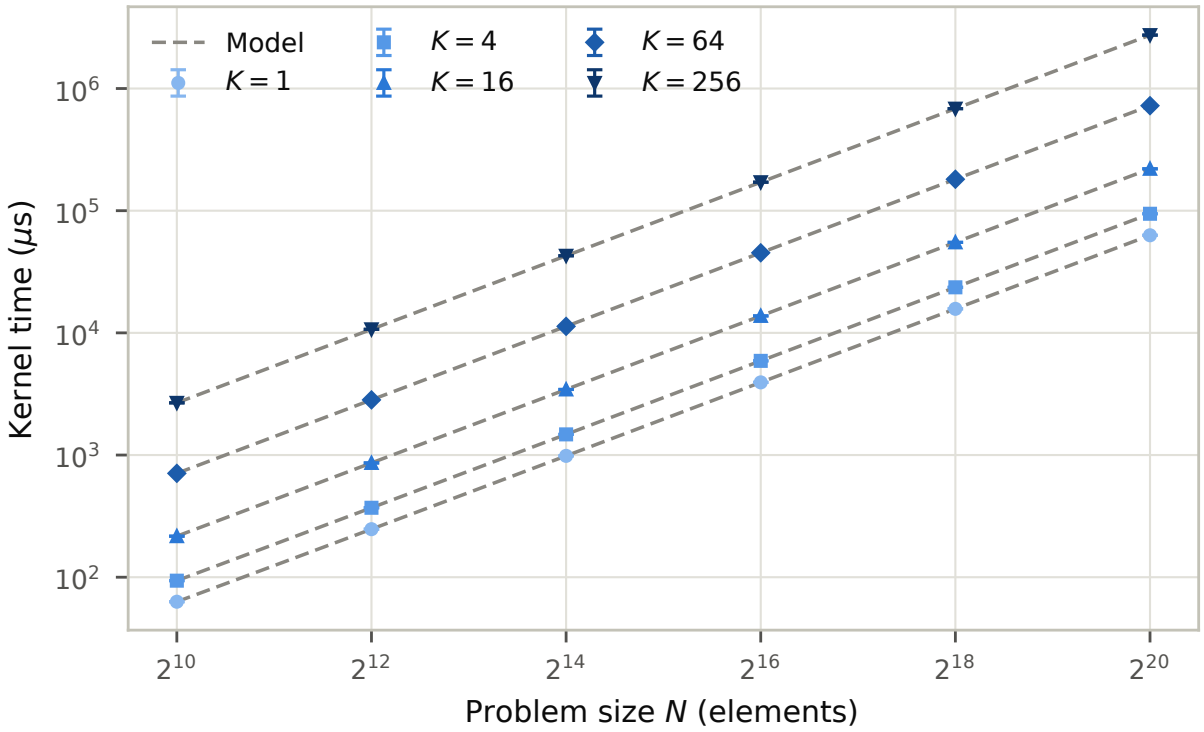


Figure 3. Kernel time of the baseline accelerator ($P = 1$) versus problem size. Markers are medians of 20 hardware runs, and error bars show ±1 standard deviation (hidden by the markers when smaller); dashed lines are the calibrated model $T = T_0 + N(K + 5)/f_{PL}$ with $T_0 = 1.55$ µs and $f_{PL} = 100$ MHz.

Given this result, it is possible to say that the baseline accelerator is limited by its schedule and not by DDR bandwidth: at one element every $K + 5$ cycles, it moves at most 8 bytes every 60 ns (about 133 MB/s at $K = 1$), far below the capability of the 128-bit HP port. The inner loop is pipelined with II = 1, but elements are processed one after another, and five cycles per element are spent outside the pipelined loop. Second, and more importantly for offloading, the baseline accelerator is slower than both processor baselines: at $N = 2^{20}$, it is 4.5–5.2 times slower than CPU-1 and 18–20 times slower than CPU-4 over the whole range of intensities (Tables 4 and 5), even before any cache-maintenance or configuration overhead is added. A pipelined inner loop is therefore not sufficient to make offloading profitable; parallelism across elements is required.

4.4. Model-Based Profitability Analysis

The calibrated parameters allow the minimum parallelism of Equation (8) to be evaluated before implementing any further design. With $\beta_1 = 1.92$ ns and $f_{PL} = 100$ MHz, one obtains $P_{min,1} \approx 5.2$: at 100 MHz, the PL must process more than about five elements

concurrently to outperform one Cortex-A53 core as the computational intensity grows. For the four-core baseline, $\beta_4 = 0.487$ ns gives $P_{\min,4} \approx 21$. Using a multicore baseline instead of a single core thus raises the required accelerator parallelism by a factor of about four. Raising the PL clock to 200 MHz would halve both values, to about 2.6 and 10 lanes. In the first campaign, interference had flattened the CPU-4 slope to 0.75 ns, which would have suggested only 13 lanes.

Table 6 and Figure 4 combine the measured processor costs with the model of Equation (5), assuming $c_P = 5$ cycles per group for all P and bounding the accelerator cost from below with the interface limit of Equation (6) for $w = 32$ bits. Under these assumptions, four lanes do not outperform either processor baseline at any intensity, and eight lanes outperform CPU-1 at all intensities but CPU-4 at none. Because the multicore baseline scales almost ideally, sixteen lanes at 100 MHz would not outperform CPU-4 at any intensity either: with a 32-bit interface, the bound of 10 ns per element exceeds the CPU-4 cost at $K \leq 4$, and with a 128-bit interface, which lowers the bound to 2.5 ns, the compute-bound cost of sixteen lanes remains 10–28% above the CPU-4 cost at every K. Doubling the PL clock changes the picture: at 200 MHz, a burst-friendly sixteen-lane design with a 128-bit interface would be 1.6–1.8 times faster than CPU-4 at every intensity (last row of Table 6). These predictions assume that the per-group overhead of a multi-lane design remains that of the baseline design; they are the working hypotheses tested with the multi-lane accelerator in Section 4.5.

Table 6. Per-element cost (ns) at $N = 2^{20}$ of the processor baselines and of accelerators with P lanes at 100 MHz unless stated otherwise. Rows marked "meas." are measured (medians of 20 runs, see Tables 4, 5 and 7 for the standard deviations); the other accelerator rows are model predictions (Equation (5) with $c_P = 5$, bounded by Equation (6) for a 32-bit interface unless 128-bit is indicated). Values in bold are lower than the measured CPU-4 cost at the same K.

Configuration	$K = 1$	$K = 4$	$K = 16$	$K = 64$	$K = 256$
CPU-1 (meas.)	13.44	17.44	42.18	141.10	502.24
CPU-4 (meas.)	3.41	4.41	10.75	35.47	128.15
Ideal 4-core (CPU-1/4)	3.36	4.36	10.55	35.28	125.56
Accelerator $P = 1$ (meas.)	60.0	90.0	210.0	690.0	2610.0
Accelerator $P = 4$, $c_P = 5$	15.0	22.5	52.5	172.5	652.5
Accelerator $P = 8$	10.0	11.25	26.25	86.25	326.25
Accelerator $P = 16$	10.0	10.0	13.13	43.13	163.13
Accelerator $P = 16$, 128-bit	3.75	5.63	13.13	43.13	163.13
Accelerator $P = 16$, 128-bit, 200 MHz	**1.88**	**2.81**	**6.56**	**21.56**	**81.56**
Accelerator $P = 4$ (meas.)	1032.3	1036.8	1075.2	1191.4	1673.2

4.5. Multi-Lane Accelerator

Table 7 reports the kernel times of the four-lane accelerator over the same grid. All configurations produced zero mismatching elements. The compute part of the model is confirmed: a least-squares fit of Equation (5) with free coefficients returns 1.005 PL cycles per inner iteration and per group of four elements, i.e., the four lanes perform four updates per cycle as designed, and the per-element slope measured at $N = 2^{20}$ (2.51 ns per iteration) matches the value $1/(Pf_{\mathrm{PL}}) = 2.5$ ns used in Table 6. The overhead term, by contrast, is far larger than assumed. Fixing the per-iteration coefficient to one cycle and T_0 to the 1.55 µs of the baseline design, the fit yields $c_4 = 412.5$ overhead cycles per group and reproduces all 30 configurations within 0.93%; the coefficient of variation across the 20 runs for each configuration is at most 0.33%. The value of c_4 is 82 times the five cycles per group assumed for the baseline accelerator in Table 6. As a consequence, the four-lane accelerator costs

1032 ns per element at $K = 1$ instead of the predicted 15 ns, and the model predicts it to be slower than the single-lane design whenever $2.5\,(K + 412) > 10\,(K + 5)$ ns, i.e., for $K \lesssim 131$. The measurements confirm this break-even point at every N: the four-lane design is 17, 11.4, 5.1 and 1.7 times slower than the baseline at $K = 1$, 4, 16 and 64, respectively, and 1.56 times faster at $K = 256$.

Table 7. Kernel time (µs, median ± standard deviation of 20 runs) of the accelerator with $P = 4$ measured on the KV260 at $f_{PL} = 100$ MHz. All runs produced zero mismatching elements. The model $T = T_0 + \lceil N/4 \rceil (K + c_4)/f_{PL}$ with $T_0 = 1.55$ µs and $c_4 = 412.50$ reproduces every median within 0.93%.

N	$K = 1$	$K = 4$	$K = 16$	$K = 64$	$K = 256$
1024	1051.47 ± 0.86	1057.99 ± 0.55	1096.16 ± 0.89	1219.91 ± 0.20	1702.17 ± 1.08
4096	4204.63 ± 2.84	4237.01 ± 1.94	4402.83 ± 1.06	4878.23 ± 0.40	6817.57 ± 22.31
16,384	16,910.22 ± 3.11	16,981.27 ± 6.05	17,613.84 ± 22.97	19,517.15 ± 1.82	27,412.21 ± 20.53
65,536	67,641.83 ± 40.88	67,926.25 ± 89.07	70,455.21 ± 28.63	78,071.91 ± 60.18	109,638.40 ± 31.46
262,144	270,564.70 ± 91.00	271,714.96 ± 175.84	281,820.21 ± 68.62	312,272.24 ± 112.63	438,597.63 ± 65.13
1,048,576	1,082,437.38 ± 84.18	1,087,181.99 ± 124.10	1,127,409.32 ± 73.97	1,249,299.74 ± 82.33	1,754,423.48 ± 60.70

The overhead of about 412 cycles per group corresponds to roughly 51 cycles (0.51 µs) for each of the eight 32-bit memory accesses of a group if these are fully serialized, a value compatible with the round-trip latency of an individual transaction from the PL to DDR memory through the SmartConnect, the HP port and the memory controller. The two designs differ precisely in how they access memory: the baseline reads and writes one element per outer iteration unconditionally, whereas the four-lane design issues four loads and four stores per group, each guarded by a boundary condition (Appendix A). The HLS synthesis logs confirm the consequence: for the baseline design, the tool reports burst reads and burst writes of variable length inferred on both bundles (message HLS 214-115), whereas for the four-lane design no burst transfer is inferred, so that every access pays the full latency of an individual AXI transaction.

Table 8 compares the four-lane accelerator with the processor baselines at $N = 2^{20}$. As K grows from 1 to 256, the accelerator is 77 to 3.3 times slower than CPU-1 and 303 to 13 times slower than CPU-4. The prediction in Table 6 that four lanes are profitable against neither baseline is therefore confirmed, but the margin is far wider than predicted, and its origin is different: the gap is dominated by per-group memory-access overhead rather than insufficient parallelism. Even with the overhead of the baseline design, four lanes would remain below $P_{\min,1} \approx 5.2$ and $P_{\min,4} \approx 21$.

Table 8. Accelerator with $P = 4$ versus processor baselines at $N = 1{,}048{,}576$ (kernel time; median ± standard deviation). The slowdown is $T_{acc}/T_{CPU} = 1/S$, so that values above 1 indicate that offloading is not profitable.

K	CPU-1 (ms)	CPU-4 (ms)	Accelerator $P = 4$ (ms)	Slowdown vs. CPU-1	Slowdown vs. CPU-4
1	14.10 ± 0.08	3.57 ± 0.71	1082.44 ± 0.08	76.80 ± 0.44	302.85 ± 60.35
4	18.29 ± 0.14	4.63 ± 0.71	1087.18 ± 0.12	59.43 ± 0.44	234.93 ± 35.82
16	44.22 ± 0.16	11.27 ± 0.88	1127.41 ± 0.07	25.49 ± 0.09	100.06 ± 7.81
64	147.95 ± 0.19	37.20 ± 1.00	1249.30 ± 0.08	8.44 ± 0.01	33.59 ± 0.90
256	526.64 ± 0.26	134.38 ± 4.03	1754.42 ± 0.06	3.33 ± <0.01	13.06 ± 0.39

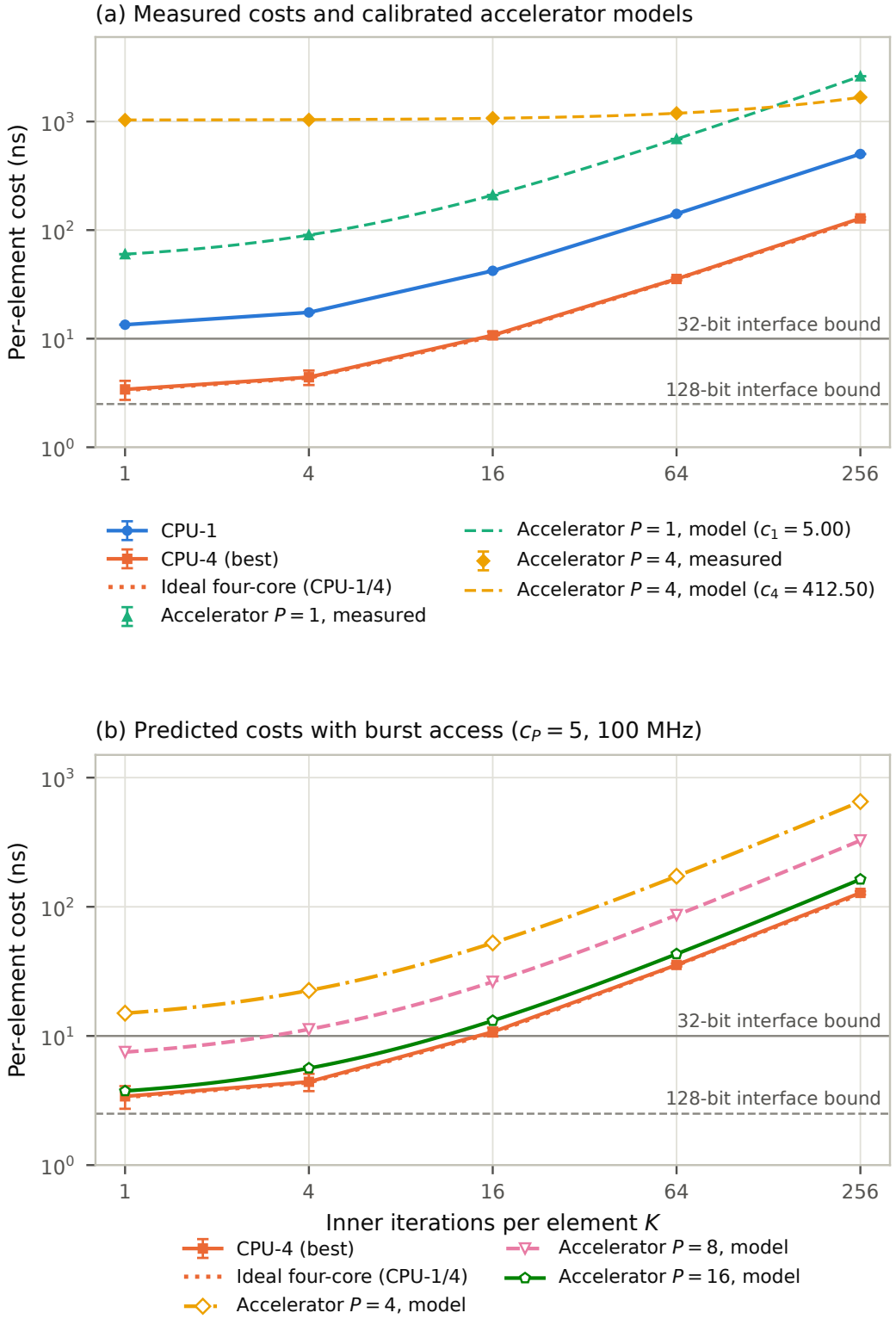


Figure 4. Per-element cost versus computational intensity at $N = 2^{20}$; measured values are medians of 20 runs with error bars of ±1 standard deviation (hidden by the markers when smaller). (**a**) Processor baselines and both accelerators, with the calibrated models of the single-lane design ($c_1 = 5$) and of the four-lane design ($c_4 = 412.5$). (**b**) Model predictions (Equation (5), $f_{\mathrm{PL}} = 100$ MHz) for accelerators with 4, 8 and 16 lanes that keep the per-group overhead of the single-lane design, compared with the four-core baseline. In both panels, the effective accelerator cost is the maximum between its compute-bound curve and the interface bound (horizontal lines).

4.6. End-to-End Offloading Cost

Table 9 decomposes the end-to-end time for $K = 1$, where the kernel is shortest and the relative weight of the other terms is largest. Programming and reading back the six argument registers takes 2.0 µs. Both synchronization requests are well described by affine functions of the synchronized size, $T_{s2d} \approx 13.6\ \mu s + 0.071\ \mu s/KiB$ over the input and output regions ($8N$ bytes) and $T_{s2c} \approx 15.6\ \mu s + 0.073\ \mu s/KiB$ over the output region ($4N$ bytes). Each invocation therefore carries a fixed cost of about 31 µs plus about 0.84 ns per element. With the host program of the first campaign, the same two requests had a fixed part of about 125 µs each, for a fixed cost of about 0.26 ms per invocation, whereas their size-dependent part was similar (0.075–0.084 µs/KiB). The cache operations themselves therefore account mainly for the size-dependent part, while the fixed part is dominated by the software path through which the requests are issued; the revised host keeps the `sysfs` attributes of the driver open for the whole run. For the baseline accelerator at $K = 1$, the time spent outside the kernel falls from 31% of T_{e2e} at $N = 1024$ to 1.4% at $N = 2^{20}$; for the four-lane design, whose kernel is 17 times slower, it falls from 2.8% to 0.08%. Because both accelerators are already slower than the processor baselines in kernel time alone, these terms only widen the gap. They remain decisive at small N: the fixed invocation cost alone exceeds the complete execution time of CPU-4 up to $N = 2^{12}$ for $K \leq 4$ and at $N = 2^{10}$ for $K = 16$.

Table 9. Decomposition of the end-to-end offloading time (Equation (3)) for $K = 1$, in µs (median ± standard deviation). T_{cfg}, T_{s2d} and T_{s2c} do not depend on the kernel and are pooled over all designs and all K. "Outside" is the fraction of T_{e2e} spent outside the kernel.

N	Data	T_{cfg}	T_{s2d}	T_{s2c}	T_{kern} $P = 1$	Outside (%)	T_{kern} $P = 4$	Outside (%)
1024	8 KiB	2.02 ± 0.59	13.33 ± 0.67	13.06 ± 1.04	63.02 ± 0.10	31.18 ± 0.83	1051.47 ± 0.86	2.77 ± 0.11
4096	32 KiB	2.02 ± 0.01	15.25 ± 0.78	15.53 ± 2.48	247.29 ± 0.09	11.12 ± 0.29	4204.63 ± 2.84	0.78 ± 0.03
16,384	128 KiB	2.02 ± 0.56	21.49 ± 1.69	19.35 ± 3.85	984.64 ± 0.09	4.02 ± 0.24	16,910.22 ± 3.11	0.26 ± 0.01
65,536	512 KiB	2.02 ± 0.54	47.41 ± 4.28	33.85 ± 6.62	3933.73 ± 0.11	1.97 ± 0.05	67,641.83 ± 40.88	0.13 ± 0.01
262,144	2 MiB	2.03 ± 0.06	164.84 ± 8.95	96.63 ± 10.34	15,730.28 ± 0.08	1.47 ± 0.04	270,564.70 ± 91.00	0.10 ± <0.01
1,048,576	8 MiB	2.03 ± 0.05	592.16 ± 8.43	310.99 ± 12.15	62,916.61 ± 0.13	1.37 ± 0.02	1,082,437.38 ± 84.18	0.08 ± <0.01

4.7. Offloading Map

Figure 5 compares the end-to-end accelerator time with the strongest processor baseline over the full grid. CPU-4 is the fastest processor configuration in every cell, and no measured accelerator outperforms it: the better accelerator, which is the baseline design for $K \leq 64$ and the four-lane design for $K = 256$, remains 13 to 22 times slower end-to-end, and the bootstrap 95% confidence intervals of the two medians do not overlap in any of the 30 configurations. The gap is nearly uniform over the grid because the fixed invocation cost is now small. Unbound CPU-4 threads no longer produce the region of apparent profitability at small N that the first campaign had shown, in which the accelerator appeared up to 7 times faster than CPU-4 at $N = 1024$ and $K = 1$; that region was therefore an artefact of the measurement conditions rather than of thread placement.

4.8. Energy

Table 10 and Figure 6 report the energy at $N = 2^{20}$. The SOM draws 3.12 W at idle with either design loaded, and this static part dominates the energy of every target. CPU-4 raises the power to 3.9–4.4 W but finishes almost four times sooner than CPU-1, and it is the most energy-efficient target at every intensity, with 17.9 to 561 nJ per element. CPU-1 needs 2.6–3.0 times more energy. The accelerators draw only 3.2–3.3 W, but they run 12 to 260 times longer than CPU-4 and therefore need 11–15 (single-lane) and 9.6–189 (four-lane) times more energy. Sleeping between polls lowers the accelerator power by about 0.05 W and its energy by about 1.5%, so the busy-polling core is not the reason for the

gap. Even the dynamic energy alone does not favour offloading: with sleep polling, the single-lane accelerator needs 1.1–1.3 times the dynamic energy of CPU-4, and with busy polling 1.9–2.3 times. CPU-1 and CPU-4 need almost the same dynamic energy (within 5%), so parallelization on the processor saves static energy at no dynamic cost. In terms of energy–delay product, the better accelerator is 112 to 225 times worse than CPU-4. On this platform, energy does not move the offloading boundary towards the accelerator but further away from it, because the static power of the module is paid for the whole, longer execution of the accelerator.

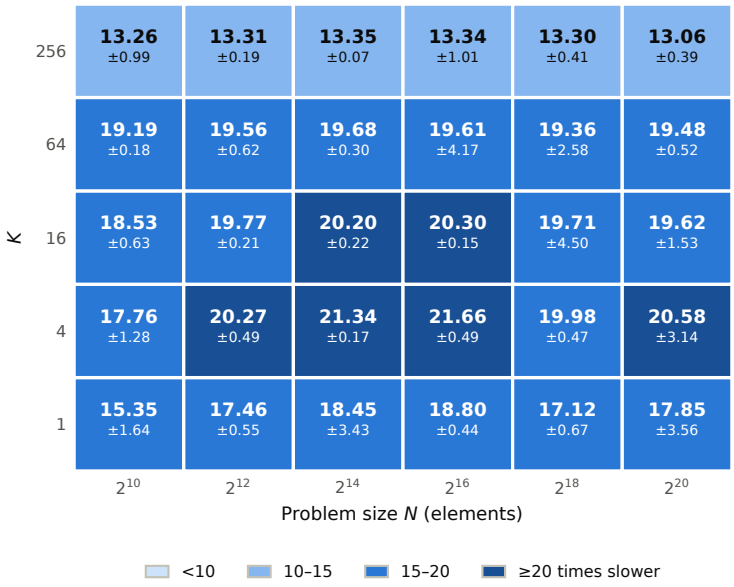


Figure 5. Measured offloading map over the (N, K) grid, based on the end-to-end accelerator time of Equation (3) and on the processor time (medians of 20 runs). The fastest configuration in every cell is CPU-4 (best measurement). The number and the shading give the end-to-end time of the faster accelerator ($P = 1$ for $K \leq 64$, $P = 4$ for $K = 256$) divided by that of CPU-4, with its standard deviation propagated from those of the two times. No cell shows an accelerator faster than CPU-4, with bound or unbound threads.

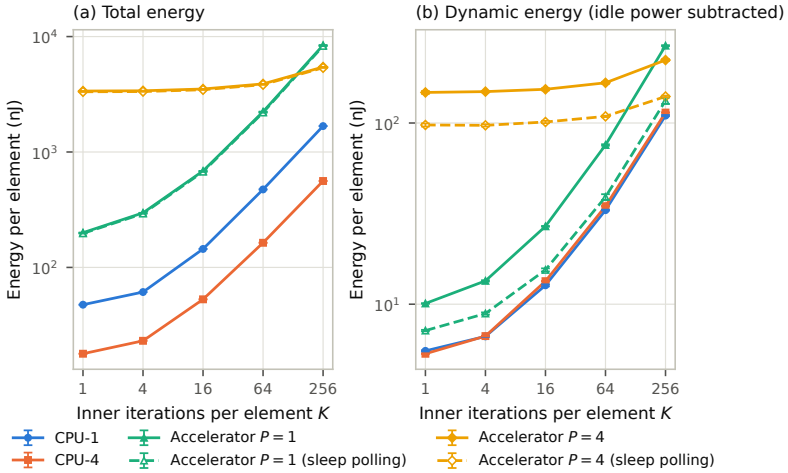


Figure 6. Energy per element at $N = 2^{20}$: mean ±1 standard deviation over three repetitions of at least 10 s each (error bars hidden by the markers when smaller). (**a**) Total energy of the SOM. (**b**) Dynamic energy, with the idle power subtracted. Hollow markers and dashed lines: accelerators with 20 µs sleeps between polls.

Table 10. Energy at $N = 2^{20}$ (mean ± standard deviation over 3 independent repetitions of at least 10 s each). Power is the mean SOM power during the measurement loop; dynamic energy subtracts the idle power with the same design loaded. Ratios refer to CPU-4 at the same K; EDP is the energy–delay product per invocation.

K	Time (ms)	Power (W)	Energy per Element (nJ)	Dynamic (nJ)	Energy vs. CPU-4	EDP vs. CPU-4
CPU-1						
1	14.11 ± 0.03	3.53 ± <0.01	47.54 ± 0.14	5.53 ± 0.05	2.66 ± 0.01	8.91 ± 0.07
4	18.32 ± 0.02	3.50 ± <0.01	61.18 ± 0.09	6.64 ± 0.04	2.65 ± 0.01	8.77 ± 0.06
16	44.29 ± 0.05	3.42 ± <0.01	144.54 ± 0.28	12.71 ± 0.17	2.73 ± 0.01	9.12 ± 0.03
64	148.10 ± 0.11	3.35 ± <0.01	473.85 ± 0.13	33.04 ± 0.21	2.90 ± 0.01	9.92 ± 0.06
256	526.38 ± 0.01	3.34 ± <0.01	1676.61 ± 0.94	109.86 ± 0.90	2.99 ± <0.01	10.50 ± 0.02
CPU-4 (bound)						
1	4.21 ± 0.01	4.45 ± 0.01	17.87 ± 0.04	5.33 ± <0.01	1.00 ± <0.01	1.00 ± 0.01
4	5.53 ± 0.02	4.39 ± <0.01	23.13 ± 0.07	6.67 ± 0.02	1.00 ± <0.01	1.00 ± 0.01
16	13.27 ± <0.01	4.18 ± <0.01	52.91 ± 0.02	13.40 ± 0.02	1.00 ± <0.01	1.00 ± <0.01
64	43.25 ± 0.13	3.96 ± <0.01	163.50 ± 0.52	34.76 ± 0.17	1.00 ± <0.01	1.00 ± 0.01
256	149.97 ± 0.13	3.92 ± <0.01	560.59 ± 0.45	114.21 ± 0.23	1.00 ± <0.01	1.00 ± <0.01
Accelerator $P = 1$						
1	63.74 ± <0.01	3.29 ± <0.01	199.93 ± 0.05	10.09 ± 0.04	11.18 ± 0.02	169.18 ± 0.90
4	95.20 ± <0.01	3.27 ± <0.01	296.99 ± 0.11	13.45 ± 0.11	12.84 ± 0.04	221.14 ± 1.42
16	221.05 ± <0.01	3.25 ± <0.01	685.24 ± 0.13	26.90 ± 0.13	12.95 ± <0.01	215.69 ± 0.11
64	724.39 ± <0.01	3.23 ± <0.01	2233.05 ± 0.70	75.64 ± 0.70	13.66 ± 0.04	228.75 ± 1.40
256	2737.67 ± <0.01	3.23 ± <0.01	8420.58 ± 1.73	267.13 ± 1.73	15.02 ± 0.01	274.21 ± 0.45
Accelerator $P = 4$						
1	1083.40 ± 0.01	3.27 ± <0.01	3373.72 ± 2.73	147.47 ± 2.69	188.74 ± 0.42	48,527.85 ± 260.26
4	1088.46 ± 0.03	3.27 ± <0.01	3390.35 ± 1.42	149.03 ± 1.34	146.58 ± 0.44	28,863.28 ± 185.81
16	1128.34 ± 0.01	3.27 ± <0.01	3513.55 ± 2.15	153.48 ± 2.14	66.40 ± 0.05	5645.29 ± 4.33
64	1250.31 ± <0.01	3.26 ± <0.01	3889.97 ± 2.15	166.67 ± 2.15	23.79 ± 0.08	687.81 ± 4.23
256	1755.45 ± 0.01	3.26 ± <0.01	5450.09 ± 2.59	222.57 ± 2.61	9.72 ± 0.01	113.80 ± 0.19

5. Discussion

5.1. Offloading Profitability (RQ1)

Across the 30 configurations, no accelerator outperformed the multicore baseline end to end; the best was 13–22 times slower and required about 10–13 times more energy. The baseline accelerator illustrates a common pitfall of HLS-based offloading: an inner loop pipelined with II = 1 does not by itself yield a high-throughput accelerator. Because elements are processed sequentially, the design achieves one inner iteration per PL cycle, i.e., 10^8 iterations per second, against about 5×10^8 for one NEON-vectorized A53 core and about 2×10^9 for four. The clock-frequency gap between the PS and the PL must therefore be compensated by spatial parallelism, and Equation (8) quantifies how much is needed. The benchmark is embarrassingly parallel, and the four-lane accelerator occupies only 2.4% of the LUTs and no DSP slices after implementation, with a positive slack of 4.96 ns at 100 MHz; sixteen to thirty-two lanes therefore appear feasible on the XCK26 in terms of resources, although timing closure and memory-interface throughput must be verified. The four-lane experiment shows, however, that lanes are necessary but not sufficient: they multiplied the compute rate as intended, yet the accelerator remained slower than the single-lane design for $K \lesssim 131$ because of how it accessed memory.

5.2. Choice of the Processor Baseline

Comparing an accelerator only against a single core would overstate its benefit on a platform that offers four cores: the multicore baseline raises the required number of PL lanes from about 5 to about 21. Our two campaigns show that the measurement conditions of the multicore baseline matter as much as its presence. In the first campaign, run without controlling background activity, the OpenMP baseline paid about 1.9 ms per invocation

with unbound threads; its repeated measurements differed by up to a factor of 13.6, and the accelerator appeared up to 7 times faster than four cores for small problems. Under controlled conditions, these effects disappeared for bound and unbound threads alike, and so did every region in which offloading seemed profitable. Because the platform has exactly four cores, any concurrent activity delays one thread and with it the whole parallel loop: even in the quiet system, the coefficient of variation of individual CPU-4 runs reached 34%, whereas their medians were reproducible within 5%. We therefore argue that offloading studies on MPSoCs should report single-core and parallel baselines running bit-identical code, control and state the background activity of the host, repeat the parallel measurements and report their dispersion, and state speedups against the best processor configuration for each problem size.

5.3. Role of Communication (RQ2)

The clearest answer to RQ2 comes from comparing the two accelerators. Moving from one to four lanes changed the memory-access pattern of the kernel; the HLS tool no longer inferred burst transfers, and the non-overlapped overhead grew from 5 to 412 cycles per group, i.e., from 5 to about 103 cycles per element. On a tightly coupled SoC, an individual 32-bit access from the PL to DDR memory that is not part of a burst costs of the order of half a microsecond at 100 MHz, two orders of magnitude more than the per-element cost of the processor at low intensity (3–13 ns at $K = 1$). A burst-friendly memory schedule—for instance, a load–compute–store organization with on-chip buffers, or a dataflow pipeline that overlaps memory accesses and computation—is therefore a prerequisite for, rather than a refinement of, parallelization in the PL. Even with such a schedule, the accelerator becomes bound by its memory interface rather than by computation at low intensity: with the 32-bit `m_axi` ports at 100 MHz, no amount of lane parallelism brings the cost below 10 ns per element, which is above both processor baselines at $K = 1$, even though the HP port itself is 128 bits wide. Finally, the invocation path adds a fixed cost of about 31 µs, almost entirely due to the two cache-maintenance requests issued through the driver, plus about 0.84 ns per element. For the baseline design, this accounts for 31% of the end-to-end time at $N = 1024$ and sets a minimum problem size below which offloading cannot pay off regardless of the accelerator design. The comparison with the first campaign, in which the same requests cost about 0.26 ms per invocation, shows that this threshold is set mostly by the host software rather than by the cache operations; lighter synchronization paths, or coherent ports, would lower it further.

5.4. Predictability (RQ3)

Three parameters described the baseline accelerator within 0.04% over 30 configurations and three orders of magnitude of N, and the measured kernel times reproduced those of the first campaign within 0.15% (within 0.001% for $N \geq 2^{16}$). This regularity, which follows from the deterministic schedule of statically scheduled HLS designs, is what makes pre-implementation reasoning about offloading boundaries possible. The four-lane design delimits what can be predicted. The compute term transferred without recalibration: the slope of 2.5 ns per iteration predicted for four lanes was measured within 1%. The overhead term did not: the uncalibrated prediction ($c_P = 5$) underestimated the cost by a factor of 69 at $K = 1$, whereas a single calibrated value ($c_4 = 412.5$) again described all 30 configurations within 0.93%. The model is therefore predictive with respect to parallelism, but the memory-access overhead must be either calibrated with one measurement or guaranteed by construction through a burst-friendly design, which is the assumption behind the predictions for eight and sixteen lanes in Table 6. Somewhat paradoxically, the least predictable component of the study was not the accelerator but the multicore processor baseline: under

uncontrolled conditions, its variability alone moved the required parallelism between 13 and 21 lanes, whereas under controlled conditions it scaled almost ideally and fixed it at about 21.

5.5. Validated Findings, Untested Predictions and Hypotheses

To separate what the experiments demonstrate from what they suggest, the main statements of this paper can be grouped by the evidence that supports them.

Findings validated by measurements on the KV260. The single-lane accelerator costs $K + 5$ PL cycles per element, with a maximum model error of 0.04% over 30 configurations and kernel times reproduced within 0.18% in two campaigns (Table 5). Four lanes multiply the compute rate by four, since the measured slope matches the prediction within 0.46%, whereas without burst transfers the per-group overhead grows from 5.00 to 412.50 cycles, as shown by the fit over 30 configurations (within 0.93%) and by the HLS synthesis logs (Section 4.5). Each invocation carries a fixed cost of 31.14 µs, dominated by the software path of the cache-maintenance requests, against 0.26 ms with the host program of the first campaign (Section 4.6). Uncontrolled background activity can distort the multicore baseline enough to create a spurious offloading region at small N, as the comparison between the first and the controlled campaign shows (Sections 3.8 and 4.2). Finally, no measured accelerator outperforms the best four-core baseline end-to-end, in any of the 30 configurations and with non-overlapping 95% confidence intervals (Section 4.7), and none is more energy-efficient than CPU-4, even in dynamic energy (Section 4.8).

Model predictions not yet tested by an implementation. According to the calibrated model (Equation (8)), at 100 MHz more than 5.21 lanes are needed to outperform one core and more than 20.52 to outperform four, and a burst-friendly sixteen-lane design with a 128-bit interface would outperform CPU-4 at every intensity only with a PL clock of 200 MHz (Table 6). Both predictions assume the per-group overhead of the baseline design, which the four-lane accelerator did not achieve.

Hypotheses beyond the scope of the experiments. That the same reasoning applies to accelerators on separate chiplets is an argument by analogy (Section 5.6). That lighter synchronization paths or coherent ports would lower the minimum profitable problem size follows from the end-to-end decomposition but was not measured. That calibrated models can drive a run-time placement policy is left to future work.

5.6. Outlook: Implications for Heterogeneous Integration

The considerations in this subsection are not tested by the experiments of this paper and should be read as hypotheses. Although this study considers a monolithic SoC, the reasoning is not specific to it. Whenever computation is moved to a specialized component, whether programmable logic on the same die or an accelerator on a separate chiplet connected by a die-to-die interface, profitability depends on the ratio between the aggregate throughput of the accelerator and the per-operation cost of the host, and on the bandwidth, latency and invocation overhead of the link [3,5]. The four-lane result suggests that the latency of an individual link transaction, and not only the link bandwidth, becomes decisive as soon as accesses are not batched; we expect the same to hold for die-to-die links, where inter-chiplet communication is already recognized as a central design concern [6,43]. Simple calibrated models of the kind used here could therefore support co-design decisions at the system and package level, for instance to determine the minimum link bandwidth, transaction size or accelerator parallelism required for a given workload class.

5.7. Limitations

This preliminary study has several limitations. A single synthetic integer kernel was used, whose regular access pattern is favourable to streaming; kernels with data reuse, irregular accesses or floating-point arithmetic may behave differently. Individual runs of the multicore baseline remained sensitive to background activity even under controlled conditions; we report medians with their standard deviations and the best of six measurements per configuration, but did not isolate cores from the operating system. The buffer was mapped as non-cacheable in user space and synchronized through the `sysfs` interface of the driver, so the measured cache-maintenance costs include system-call overheads and depend on the host software, as the comparison between the two campaigns shows. Only the non-coherent `S_AXI_HP0_FPD` port and a 100 MHz PL clock were considered, although the post-implementation slack of both designs suggests that about 200 MHz is attainable; since $P_{\min}$ is inversely proportional to f_{PL}, a higher clock would lower the required parallelism accordingly. The kernel time was measured on the host through register polling. The only multiplication of the kernel is by a constant and was mapped to logic, so that no DSP slices were used; kernels whose arithmetic maps onto DSP slices, such as filters, convolutions or neural-network layers, may move the boundary in favour of the accelerator. Finally, energy was measured at the supply of the SOM with a 50 Hz power monitor; it includes the static power of the whole module and does not separate the contributions of the PS, the PL and the memory.

5.8. Future Work

Future work will first implement a burst-friendly multi-lane accelerator with 16 to 32 lanes and wider memory interfaces to test the predictions of Table 6, and will measure the multicore baseline on isolated cores. It will then extend the characterization to coherent ports (HPC and ACP) [17,18,31], dataflow designs that overlap memory access and computation, higher PL clock frequencies, per-rail energy measurements and application kernels such as matrix multiplication, convolutions, stencils, streaming DSP filters and edge-AI layers, and use the calibrated model as the basis of a run-time policy that selects the execution target of each kernel invocation.

6. Conclusions

This paper presented a preliminary, communication-aware characterization of CPU–FPGA offloading on the AMD Kria KV260. Using a parametric microbenchmark with independently tunable problem size and computational intensity, executed bit-identically on one and four Cortex-A53 cores and in the programmable logic, we showed that a baseline HLS accelerator behaves deterministically, with a cost of $K + 5$ PL cycles per element captured within 0.04% by a three-parameter model over 30 configurations, and that it is about five times slower than one core despite a fully pipelined inner loop. Combining the calibrated models yields a simple profitability condition: at 100 MHz, the programmable logic must process more than about five elements concurrently to outperform a single core and about 21 to outperform four cores, while a 32-bit memory interface prevents any design from winning at the lowest intensities. A four-lane accelerator confirmed the predicted compute scaling exactly, but the loss of burst transfers raised its overhead from 5 to 412 cycles per group. Under controlled conditions, every measured accelerator remained 13 to 22 times slower than four Cortex-A53 cores end to end and needed about 10 to 13 times more energy, because the static power of the module is paid for the whole, longer accelerator execution. The fixed invocation cost of about 31 µs was dominated by the software path of the cache-maintenance requests, and the only region in which offloading appeared profitable, observed in a first campaign, was an artefact of uncontrolled measurement conditions. In practice, a straightforward HLS description is not sufficient to outperform well-used

processor cores: accelerator parallelism, memory schedule and interface width must be designed together. These results suggest that lightweight, calibrated models can anticipate offloading boundaries on tightly coupled heterogeneous SoCs and can guide the sizing of accelerators and interfaces before implementation, provided that memory-access and invocation overheads are treated as first-order design parameters and that the processor baseline is chosen, configured, measured under controlled conditions and reported with its dispersion.

Author Contributions: Conceptualization, R.B., A.S. and L.V.; methodology, R.B. and L.V.; software, R.B.; validation, R.B. and A.S.; formal analysis, R.B.; investigation, R.B.; resources, L.V. and D.P.P.; data curation, R.B.; writing—original draft preparation, R.B.; writing—review and editing, A.S. and L.V.; visualization, R.B.; supervision, A.S., D.P.P. and L.V.; project administration, L.V. and D.P.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The measurement logs, HLS sources, Vivado block-design scripts and host programs supporting the reported results are publicly available at https://github.com/sab-anna/Offloading-Boundaries-CPU-FPGA.git, together with the bitstreams and the scripts that automate the measurements.

Acknowledgments: Generative AI tools (Claude Opus 5.5) were used to assist with the organization of the literature search and the drafting of parts of the manuscript. All measurements were performed by the authors on the physical hardware, and all generated text, code and analyses were reviewed and verified by the authors, who take full responsibility for the content.

Conflicts of Interest: Author Danilo Pietro Pau was employed by the company STMicroelectronics. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACP	Accelerator coherency port
APU	Application processing unit
AXI	Advanced eXtensible Interface
CPU	Central processing unit
DDR	Double data rate (memory)
DMA	Direct memory access
DSP	Digital signal processing (slice)
FPGA	Field-programmable gate array
FPSoC	Field-programmable system-on-chip
GPU	Graphics processing unit
HLS	High-level synthesis
HP	High-performance (port)
II	Initiation interval
LUT	Look-up table
MPSoC	Multiprocessor system-on-chip
PL	Programmable logic
PS	Processing system
RTL	Register-transfer level
SoC	System-on-chip
SOM	System-on-module

Appendix A. HLS Kernel Sources

Listing of the baseline accelerator ($P = 1$). The processor baselines use the same loop nest; CPU-4 adds `#pragma omp parallel for` to the outer loop.

```
#include <stdint.h>
extern "C" {
 void microbench(const uint32_t *in, uint32_t *out, int n, int k) {
  #pragma HLS INTERFACE m_axi port=in  offset=slave bundle=gmem0
  #pragma HLS INTERFACE m_axi port=out offset=slave bundle=gmem1
  #pragma HLS INTERFACE s_axilite port=in
  #pragma HLS INTERFACE s_axilite port=out
  #pragma HLS INTERFACE s_axilite port=n
  #pragma HLS INTERFACE s_axilite port=k
  #pragma HLS INTERFACE s_axilite port=return
  for (int i = 0; i < n; ++i) {
   uint32_t x = in[i];
   uint32_t acc = 0;
   for (int j = 0; j < k; ++j) {
    #pragma HLS PIPELINE II=1
    uint32_t t = (x ^ (uint32_t)j) * 1664525u + 1013904223u;
    acc += t;
   }
   out[i] = acc;
  }
 }
}
```

Listing of the multi-lane accelerator ($P = 4$) as used in C simulation, C/RTL co-simulation and in the hardware experiments.

```
void microbench_p4(const uint32_t *in, uint32_t *out, int n, int k) {
 /* same INTERFACE pragmas as above, with depth=2048 for co-simulation */
 const int P = 4;
 for (int base = 0; base < n; base += P) {
  uint32_t x[P], acc[P];
  #pragma HLS ARRAY_PARTITION variable=x complete
  #pragma HLS ARRAY_PARTITION variable=acc complete
  for (int p = 0; p < P; ++p) {
   #pragma HLS UNROLL
   x[p] = (base + p < n) ? in[base + p] : 0;
   acc[p] = 0;
  }
  for (int j = 0; j < k; ++j) {
   #pragma HLS PIPELINE II=1
   for (int p = 0; p < P; ++p) {
    #pragma HLS UNROLL
    acc[p] += (x[p] ^ (uint32_t)j) * 1664525u + 1013904223u;
   }
  }
  for (int p = 0; p < P; ++p) {
   #pragma HLS UNROLL
   if (base + p < n) out[base + p] = acc[p];
  }
 }
}
```

References

1. Esmaeilzadeh, H.; Blem, E.; St. Amant, R.; Sankaralingam, K.; Burger, D. Dark Silicon and the End of Multicore Scaling. In Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA), San Jose, CA, USA, 4–8 June 2011; pp. 365–376. [CrossRef]
2. Hennessy, J.L.; Patterson, D.A. A New Golden Age for Computer Architecture. *Commun. ACM* **2019**, *62*, 48–60. [CrossRef]
3. IEEE Electronics Packaging Society. Heterogeneous Integration Roadmap, 2024 Edition. Available online: https://eps.ieee.org/technology/heterogeneous-integration-roadmap/2024-edition/ (accessed on 4 November 2026).
4. Wong, H. Launching the *Semiconductors and Heterogeneous Integration* Journal. *Semicond. Heterog. Integr.* **2026**, *1*, 1. [CrossRef]
5. Li, T.; Hou, J.; Yan, J.; Liu, R.; Yang, H.; Sun, Z. Chiplet Heterogeneous Integration Technology—Status and Challenges. *Electronics* **2020**, *9*, 670. [CrossRef]

6. Chen, S.; Zhang, H.; Ling, Z.; Zhai, J.; Yu, B. The Survey of Chiplet-Based Integrated Architecture: An EDA Perspective. In Proceedings of the 30th Asia and South Pacific Design Automation Conference (ASP-DAC), Tokyo, Japan, 20–23 January 2025.
7. Hill, M.D.; Marty, M.R. Amdahl's Law in the Multicore Era. *Computer* **2008**, *41*, 33–38. [CrossRef]
8. Gregg, C.; Hazelwood, K. Where Is the Data? Why You Cannot Debate CPU vs. GPU Performance Without the Answer. In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Austin, TX, USA, 10–12 April 2011; pp. 134–144. [CrossRef]
9. Advanced Micro Devices, Inc. *Zynq UltraScale+ Device Technical Reference Manual (UG1085)*; AMD: Santa Clara, CA, USA, 2026. Available online: https://docs.amd.com/r/en-US/ug1085-zynq-ultrascale-trm (accessed on 30 September 2026).
10. Advanced Micro Devices, Inc. *Kria K26 SOM Data Sheet (DS987)*, v1.6; AMD: Santa Clara, CA, USA, 2026. Available online: https://docs.amd.com/r/en-US/ds987-k26-som (accessed on 30 September 2026).
11. Advanced Micro Devices, Inc. *Kria KV260 Vision AI Starter Kit User Guide (UG1089)*; AMD: Santa Clara, CA, USA, 2025. Available online: https://docs.amd.com/r/en-US/ug1089-kv260-starter-kit (accessed on 30 September 2026).
12. Cong, J.; Liu, B.; Neuendorffer, S.; Noguera, J.; Vissers, K.; Zhang, Z. High-Level Synthesis for FPGAs: From Prototyping to Deployment. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2011**, *30*, 473–491. [CrossRef]
13. Nane, R.; Sima, V.M.; Pilato, C.; Choi, J.; Fort, B.; Canis, A.; Chen, Y.T.; Hsiao, H.; Brown, S.; Ferrandi, F.; et al. A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2016**, *35*, 1591–1604. [CrossRef]
14. Cong, J.; Lau, J.; Liu, G.; Neuendorffer, S.; Pan, P.; Vissers, K.; Zhang, Z. FPGA HLS Today: Successes, Challenges, and Opportunities. *ACM Trans. Reconfigurable Technol. Syst.* **2022**, *15*, 51. [CrossRef]
15. Advanced Micro Devices, Inc. *Vitis High-Level Synthesis User Guide (UG1399)*; AMD: Santa Clara, CA, USA, 2026. Available online: https://docs.amd.com/r/en-US/ug1399-vitis-hls (accessed on 30 September 2026).
16. Göbel, M.; Elhossini, A.; Chi, C.C.; Alvarez-Mesa, M.; Juurlink, B. A Quantitative Analysis of the Memory Architecture of FPGA-SoCs. In *Applied Reconfigurable Computing (ARC 2017)*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2017; Volume 10216, pp. 241–252. [CrossRef]
17. Sadri, M.; Weis, C.; Wehn, N.; Benini, L. Energy and Performance Exploration of Accelerator Coherency Port Using Xilinx ZYNQ. In Proceedings of the 10th FPGAworld Conference, Stockholm, Sweden, 10–12 September 2013. [CrossRef]
18. Min, S.W.; Huang, S.; El-Hadedy, M.; Xiong, J.; Chen, D.; Hwu, W.m. Analysis and Optimization of I/O Cache Coherency Strategies for SoC-FPGA Device. In Proceedings of the 29th International Conference on Field Programmable Logic and Applications (FPL), Barcelona, Spain, 9–13 September 2019.
19. Molanes, R.F.; Costas, L.; Rodríguez-Andina, J.J.; Fariña, J. Comparative Analysis of Processor–FPGA Communication Performance in Low-Cost FPSoCs. *IEEE Trans. Ind. Inform.* **2021**, *17*, 3826–3835. [CrossRef]
20. Williams, S.; Waterman, A.; Patterson, D. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* **2009**, *52*, 65–76. [CrossRef]
21. da Silva, B.; Braeken, A.; D'Hollander, E.H.; Touhafi, A. Performance Modeling for FPGAs: Extending the Roofline Model with High-Level Synthesis Tools. *Int. J. Reconfigurable Comput.* **2013**, *2013*, 428078. [CrossRef]
22. Siracusa, M.; Del Sozzo, E.; Rabozzi, M.; Di Tucci, L.; Williams, S.; Sciuto, D.; Santambrogio, M.D. A Comprehensive Methodology to Optimize FPGA Designs via the Roofline Model. *IEEE Trans. Comput.* **2022**, *71*, 1903–1915. [CrossRef]
23. Altaf, M.S.B.; Wood, D.A. LogCA: A High-Level Performance Model for Hardware Accelerators. In Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), Toronto, ON, Canada, 24–28 June 2017; pp. 375–388. [CrossRef]
24. Colagrande, L.; Benini, L. Optimizing Offload Performance in Heterogeneous MPSoCs. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE), Valencia, Spain, 25–27 March 2024. [CrossRef]
25. Choi, Y.k.; Cong, J.; Fang, Z.; Hao, Y.; Reinman, G.; Wei, P. A Quantitative Analysis on Microarchitectures of Modern CPU-FPGA Platforms. In Proceedings of the 53rd Annual Design Automation Conference (DAC), Austin, TX, USA, 5–9 June 2016. [CrossRef]
26. Choi, Y.k.; Cong, J.; Fang, Z.; Hao, Y.; Reinman, G.; Wei, P. In-Depth Analysis on Microarchitectures of Modern Heterogeneous CPU-FPGA Platforms. *ACM Trans. Reconfigurable Technol. Syst.* **2019**, *12*, 4. [CrossRef]
27. Molanes, R.F.; Rodríguez-Andina, J.J.; Fariña, J. Performance Characterization and Design Guidelines for Efficient Processor–FPGA Communication in Cyclone V FPSoCs. *IEEE Trans. Ind. Electron.* **2018**, *65*, 4368–4377. [CrossRef]
28. Belloch, J.A.; León, G.; Badía, J.M.; Lindoso, A.; San Millan, E. Evaluating the Computational Performance of the Xilinx Ultrascale+ EG Heterogeneous MPSoC. *J. Supercomput.* **2021**, *77*, 2124–2137. [CrossRef]
29. Bolat, A.; Siddiqui, F.; Sezer, S.; Tasdemir, K.; Khan, R. Investigation of Communication Overhead of SoC Lookaside Accelerators. In Proceedings of the 36th IEEE International System-on-Chip Conference (SOCC), Santa Clara, CA, USA, 5–8 September 2023. [CrossRef]
30. Kroh, A.; Diessel, O. A Short-Transfer Model for Tightly-Coupled CPU-FPGA Platforms. In Proceedings of the 2018 International Conference on Field-Programmable Technology (FPT), Naha, Japan, 10–14 December 2018; pp. 369–372. [CrossRef]

31. Micallef, E.; Xiao, Y.; DeHon, A. HLS-Compatible, Embedded-Processor Stream Links. In Proceedings of the 29th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Orlando, FL, USA, 9–12 May 2021.
32. Weisz, G.; Melber, J.; Wang, Y.; Fleming, K.; Nurvitadhi, E.; Hoe, J.C. A Study of Pointer-Chasing Performance on Shared-Memory Processor-FPGA Systems. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), Monterey, CA, USA, 21–23 February 2016; pp. 264–273. [CrossRef]
33. Berrazueta-Mena, D.; Navas, B. AHA: Design and Evaluation of Compute-Intensive Hardware Accelerators for AMD-Xilinx Zynq SoCs Using HLS IP Flow. *Computers* **2025**, *14*, 189. [CrossRef]
34. Li, R.; Chen, S. Design and Implementation of an FPGA-Based Tiled Matrix Multiplication Accelerator for Transformer Self-Attention on the Xilinx KV260 SoM. *arXiv* **2025**, arXiv:2503.16731.
35. Costa, C.; Brito, J.H. Throughput Impact of Software Multithreading for Deep-Learning Inference on the AMD Kria KV260. *J. Real-Time Image Process.* **2026**, *23*, 89. [CrossRef]
36. Cota, E.G.; Mantovani, P.; Di Guglielmo, G.; Carloni, L.P. An Analysis of Accelerator Coupling in Heterogeneous Architectures. In Proceedings of the 52nd Annual Design Automation Conference (DAC), San Francisco, CA, USA, 7–11 June 2015; pp. 1–6. [CrossRef]
37. Shao, Y.S.; Xi, S.L.; Srinivasan, V.; Wei, G.Y.; Brooks, D. Co-Designing Accelerators and SoC Interfaces Using gem5-Aladdin. In Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, 15–19 October 2016. [CrossRef]
38. Huang, S.; Chang, L.W.; El Hajj, I.; Garcia de Gonzalo, S.; Gómez-Luna, J.; Chalamalasetti, S.R.; El-Hadedy, M.; Milojicic, D.; Mutlu, O.; Chen, D.; et al. Analysis and Modeling of Collaborative Execution Strategies for Heterogeneous CPU-FPGA Architectures. In Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering (ICPE), Mumbai, India, 7–11 April 2019. [CrossRef]
39. Kroh, A.; Diessel, O. Efficient Fine-Grained Processor-Logic Interactions on the Cache-Coherent Zynq Platform. *ACM Trans. Reconfigurable Technol. Syst.* **2019**, *11*, 1–22. [CrossRef]
40. Kawazome, I. u-dma-buf: User Space Mappable DMA Buffer. Available online: https://github.com/ikwzm/udmabuf (accessed on 30 September 2026).
41. Press, W.H.; Teukolsky, S.A.; Vetterling, W.T.; Flannery, B.P. *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed.; Cambridge University Press: Cambridge, UK, 1992.
42. Dagum, L.; Menon, R. OpenMP: An Industry Standard API for Shared-Memory Programming. *IEEE Comput. Sci. Eng.* **1998**, *5*, 46–55. [CrossRef]
43. Das Sharma, D.; Choudhary, S.; Onufryk, P.Z.; Pelt, R. On-Package Memory with Universal Chiplet Interconnect Express (UCIe): A Low Power, High Bandwidth, Low Latency and Low Cost Approach. In Proceedings of the IEEE Symposium on High-Performance Interconnects (HOTI), San Jose, CA, USA, 20–22 August 2025; pp. 87–96.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.