RedMule: A mixed-precision matrix–matrix operation engine for flexible and energy-efficient on-chip linear algebra and TinyML training acceleration

RedMule: A Mixed-Precision Matrix-Matrix Operation Engine for Flexible and Energy-Efficient On-Chip Linear Algebra and TinyML Training Acceleration

Yvan Tortorella[a], Luca Bertaccini[b], Luca Benini[a,b], Davide Rossi[a], Francesco Conti[a]

[a] *University of Bologna, Bologna, Italy*
[b] *ETH Zurich, Zurich, Switzerland*

Abstract

The increasing interest in TinyML, i.e., near-sensor machine learning on power budgets of a few tens of mW, is currently pushing toward enabling TinyML-class training as opposed to inference only. Current training algorithms, based on various forms of error and gradient backpropagation, rely on floating-point matrix operations to meet the precision and dynamic range requirements. So far, the energy and power cost of these operations has been considered too high for TinyML scenarios. This paper addresses the open challenge of near-sensor training on a few mW power budget and presents RedMulE - Reduced-Precision Matrix Multiplication Engine, a low-power specialized accelerator conceived for multi-precision floating-point General Matrix-Matrix Operations (GEMM-Ops) acceleration, supporting FP16, as well as hybrid FP8 formats, with $\{sign, exponent, mantissa\} = (\{1, 4, 3\}, \{1, 5, 2\})$. We integrate RedMule into a Parallel Ultra-Low-Power (PULP) cluster containing eight energy-efficient RISC-V cores sharing a tightly-coupled data memory and implement the resulting system in a 22 nm technology. At its best efficiency point (@ 470 MHz, 0.65 V), the RedMulE-augmented PULP cluster achieves 755 GFLOPS/W and 920 GFLOPS/W during regular General Matrix-Matrix Multiplication (GEMM), and up to 1.19 TFLOPS/W and 1.67 TFLOPS/W when executing GEMM-Ops, respectively, for FP16 and FP8 input/output tensors. In its best performance point (@ 613 MHz, 0.8 V), RedMulE achieves up to 58.5 GFLOPS and 117 GFLOPS for FP16 and FP8, respectively, with 99.4% utilization of the array of Computing Elements and consuming less than 60 mW on average, thus enabling on-device training of deep learning models in TinyML application scenarios while retaining the flexibility to tackle other classes of common linear algebra problems efficiently.

Keywords: General Matrix-Matrix Multiplication, General Matrix-Matrix Operations, Hardware Accelerator, Embedded-Systems, Online-Learning, TinyML.

1. Introduction

In the last few years, the number of Internet of Things (IoT) devices connected and executing Machine Learning (ML) and, in particular, Deep Learning (DL) based algorithms such as Deep Neural Networks (DNNs) increased considerably. To reduce the amount of data sent over the network, improve energy efficiency, and prevent network congestion, the computation has been moved increasingly from data centers to energy-efficient IoT end-nodes with low power budgets (a few mW average, a hundred mW peak) [1], giving rise to the Tiny-ML field of research and application.

Extreme-edge applications like training and inference of Neural Networks (NNs), graph analysis and manipulation [2, 3], short-distance problems [4], and model-based control rely on General Matrix-Matrix Multiplications (GEMMs) or General Matrix-Matrix Operations (GEMM-Ops) as the most significant kernel. GEMM-Ops are operations that share the same structure of a GEMM but replace the canonical multiply/add with other mapping and reduction operations [5]. Due to the similarity of these computational patterns, it has recently been proposed [6] to augment TensorCores with GEMM-Ops support, thereby extending their acceleration capabilities to a broader class of applications. There is not yet an equal contribution targeting ultra-low-power embedded systems.

In desktop, mobile, and data center computing, single and double-precision Floating-Point (FP) operations are typically employed for DL and linear algebra applications, providing high accuracy at an acceptable area and energy cost. However, on embedded devices, power and area constraints are much tighter. Recently, a significant effort has gone into adapting linear algebra-based algorithms as well as online learning [7] to low-precision formats, such as FP16 [8, 9] and FP8 [10, 11], while incurring in little accuracy loss. These algorithmic advancements enabled performance and energy efficiency gains [12, 13], opening the way for deploying continual learning and adaptation of DL models on extreme-edge computing systems such as smart wearable devices. However, the computational capabilities of microcontroller units (MCUs), typically used in these devices, are minimal, especially concerning the execution of FP operations.

In this paper, we present RedMulE (Reduced-precision matrix Multiplication Engine), the first TinyML-class open-

Email addresses: yvan.tortorella@unibo.it (Yvan Tortorella), lbertaccini@iis.ee.ethz.ch (Luca Bertaccini), lbenini@iis.ee.ethz.ch (Luca Benini), davide.rossi@unibo.it (Davide Rossi), f.conti@unibo.it (Francesco Conti)

source hardware accelerator supporting mixed FP precision (Hybrid-FP8 and FP16) linear algebra that is tightly integrated within RISC-V-based Parallel Ultra-Low-Power (PULP) clusters. RedMule is highly parametric, supporting different configurations tunable at design time. It shares an L1 memory with the cluster cores, thus allowing fine-grained cooperation between the hardware accelerator and the cores. RedMulE accelerates low-precision GEMMs (the key kernels behind NN training algorithms), enabling on-chip learning on Tiny-ML-class ultra-low-power System on-Chip (SoC). Furthermore, it supports a broader family of operations called GEMM-Ops [6], thus targeting an extensive set of applications.

We prototyped our design within an 8-core PULP cluster in 22 nm CMOS technology, instantiating a RedMulE instance with 48 internal Computing Elements (CEs). RedMulE occupies only 0.15 mm^2, accounting for 24% of the entire cluster area. It achieves up to 15× speedup during regular FP16 GEMMs and up to 62× during GEMM-Ops compared to parallel execution on the RISC-V cores, reaching up to 58.5 GFLOPS (99.4% CEs utilization) at 613 MHz and 0.8 V. In its best efficiency point, i.e. 470 MHz at 0.65 V, RedMulE achieves up to 772 GFLOPS/W and 1.19 TFLOPS/W energy efficiency for GEMM and GEMM-Ops respectively, while reaching 44.8 GFLOPS. When computing on FP8 input/output tensors representation, a 96 CEs RedMulE implementation reaches up to 117 GFLOPS at 613 MHz and 0.8 V, achieving up to 920 GFLOPS/W, an energy efficiency comparable to INT8 inference accelerators.

2. Related Work

The strong interest in executing linear algebra-based algorithms like inference and training of NNs led to the development of various hardware platforms specialized in this task, spanning from data-centers computing systems to ultra-low-power embedded platforms [14]. NVIDIA's recent Hopper H100 [15] Graphic Processing Unit (GPU) is the most representative example of data-center computing platform for DL tasks like inference and training of NNs. The H100 achieves 1978 TFLOPS at 700 W power consumption and can be used to train huge NN models like transformers by using narrow FP8 formats.

On the other hand, enabling the execution of DL-based algorithms on ultra-low-power TinyML SoCs for extreme-edge devices such as smart wearable systems is challenging due to the strict power, energy, and cost constraints imposed. Extreme-edge inference is achievable in practical cases since it can be performed employing low-precision integer arithmetic, which reduces the model's memory footprint and increases the energy efficiency of the underlying architecture with a limited accuracy loss [16, 17]. On the contrary, extreme-edge NNs training faces large memory requirements and the need for FP calculations, which typically leads to power envelopes exceeding the TinyML constraints [17, 18]. In this section, we focus on embedded platforms emphasizing edge training at moderate power.

2.1. Inference Accelerators

Hardware accelerators specialized for low-power DL inference provide attractive alternatives to software-based executions [17, 19]. Diana [20], a low-power NN SoC, features a digital NN inference accelerator and an analog in-memory-computing core integrated within a shared memory subsystem working only with narrow integer formats. DNPU [21] is a fully-digital energy-efficient DL processor for convolutional and recursive NN inference acceleration designed in 65 nm technology and based on a heterogeneous architecture supporting 16-bit fixed-point arithmetic. Gemmini [22] is a 16×16 systolic accelerator designed for inference of deep NNs with 8-bit multiply-accumulate units with runtime-programmable weight stationary and output stationary dataflows.

2.2. On-Device Learning

On-device learning is an emerging and open challenge concerning training DL models on ultra-low-power general-purpose microcontrollers. To reach this aim, many works investigated algorithms like direct feedback alignment or equilibrium propagation. However, such methods have been demonstrated to be less effective than the classical backpropagation method due to severe convergence difficulties [23]. TinyOL [24] and [25] focus on training NNs using the low-budget Arduino Nano microcontroller based on Cortex-M core. On the other hand, PULP Trainlib [26], Cioflan *et al.* [27], and Ravaglia *et al.* [28] are all examples of approaches to enable on-device learning and adaptation on RISC-V multi-core PULP clusters like Vega [29], that provide mixed FP precision capabilities, spanning from IEEE 754 Standard FP32 and FP16 to *bfloat*. However, the low speed and number of available floating point units typical of ultra-low-power microcontrollers limit the performance of these libraries.

2.3. Training Accelerators

To address the limited training performance achievable by software libraries running on low-power processors, several researchers turned to hardware acceleration [14].

Cambricon-Q [30] is a training-oriented chip for high accuracy and energy efficiency based on 8-bit fixed-point arithmetic. However, many common training algorithms require floating-point operations to ensure convergence [31, 32, 33, 34]. Most training-oriented chips employing FP arithmetic are all characterized by power envelopes unsuitable for extreme-edge applications. IBM proposes [35, 36], an AI computing platform featuring 8×8 mixed-precision engines supporting FP16 and hybrid FP8 training, while [37] support only FP16 and FP32. Similarly, LNPU [38] supports mixed 8-bit and 16-bit FP precision for on-chip training. While these chips consume significantly less power than data-center GPUs during NN training (i.e. a few Watts as opposed to hundreds of Watts), they still do not meet the tens of mW power constraints of TinyML devices.

Recently, a few training-oriented SoCs that fit the power budget of extreme-edge applications have been presented. T-PIM [39] is a processing-in-memory accelerator in 28 nm technology for on-device learning. It reaches up to 250 GOPS/W

Table 1: Set of General Matrix-Matrix Operations supported by RedMulE

$\mathbf{Z} = (\mathbf{X} \circ \mathbf{W}) \star \mathbf{Y}$				
Group	**Kernel**	$\circ$	$\star$	**Res**
	Matmul	$\times$	$+$	$Z = (X \times W) + Y$
Group 1	Maximum Critical Path	$+$	max	$Z = max[Y, (X + W)]$
	All-Pairs Shortest Paths	$+$	min	$Z = min[Y, (X + W)]$
	Maximum Reliability Path	$\times$	max	$Z = max[Y, (X \times W)]$
	Minimum Reliability Path	$\times$	min	$Z = min[Y, (X \times W)]$
Group 2	Minimum Spanning Tree	max	min	$Z = min[Y, max(X, W)]$
	Maximym Capacity Path	min	max	$Z = max[Y, min(X, W)]$

during training with 0% of sparsity and within a power envelope of 51.23 mW at 280 MHz operating frequency. However, T-PIM and all the recently proposed PIM approaches do not support FP computations and are not suitable for standard back-propagation. To support NNs training at reduced power budgets, many training-oriented chips extensively employ pruning to increase sparsity during training [40], lacking generality. For example, TSUNAMI [41] and Trainer [42] are both accelerators designed for extreme-edge NN inference and training, meeting the TinyML power constraints by employing pruning and zero skipping. Anders *et al.* [43] propose a reconfigurable accelerator for dense-sparse matrix multiplications for mixed-precision computations, suitable for training-oriented applications since it features FP16 multiplications and FP32 accumulations with low area occupation and high energy efficiency. However, such an accelerator is not parametric, thus not allowing a fast scale-up at design time when higher performance is needed. In addition, its integration into a real system has not been evaluated, and it does not support compressed FP8 input/output tensors, which allows for training larger NN models on edge devices where the memory resources are limited.

2.4. GEMM-Ops Chips

All examples of training and inference-oriented chips mentioned so far target only the most common DL operations (such as matrix multiplications and convolutions). However, a large set of kernels share the same computational structure as GEMM but do not rely on multiplication and addition as elementary operations, falling into the GEMM-Ops scope. Graph analytics, such as breadth-first search [2, 3], short-distance problems [4] that are commonly used for path planning optimization in embedded drones navigation [44], and minimum spanning tree, used for computer vision [45], are examples of applications that make use of GEMM-Ops. SIMD^2 [6] addresses this issue by building functional units for GEMM-Op acceleration on top of NVIDIA Streaming Multiprocessor architecture, resembling the TensorCores structure and providing dedicated ISA extensions. The design is implemented in 45 nm technology. Adding all the SIMD^2 extensions to the baseline matrix multiplication unit results in up to 15.8× speedup with respect to executing the same kernel on CUDA cores at the cost of 69% of area overhead.

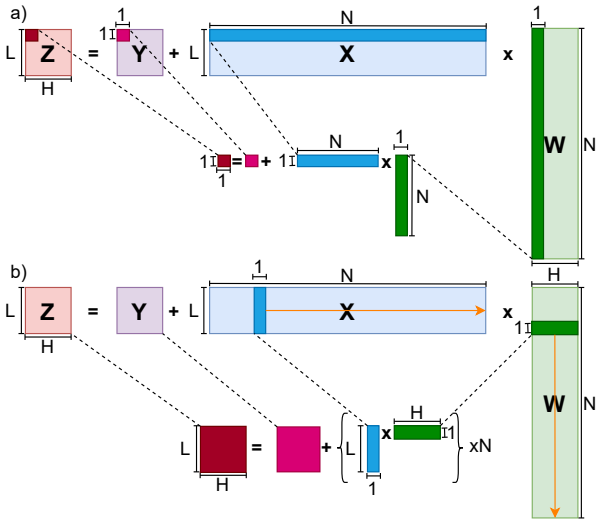


Figure 1: Execution of a GEMM through a) scalar dot product microkernel and b) block-dot product (or outer product) microkernel.

In this paper, we propose an extended version of RedMulE [46] with the following unique combination of features:

- An array of Floating-Point Units-based Computing Elements (CEs) for efficient training and inference of general DL models on embedded SoCs with additional support for reduced bit-width FP computation. We tightly couple RedMulE with a parallel cluster of RISC-V processors to achieve maximum flexibility in implementing complex training algorithms;
- Supports for GEMM-Ops with a low area overhead (16%) with respect to a GEMM-only implementation to address a wider spectrum of applications;
- A fully-parametric design that allows the instantiation of a wide range of CEs arrays, internal buffers and memory interface configurations.

3. Background

3.1. Generalized Matrix-Matrix Operations

In this work, we define *Generalized Matrix-Matrix Operations (GEMM-Ops)* as all the operations of the kind $f2(\mathbf{Y}, f1(\mathbf{X}, \mathbf{W}))$, in particular they can be expressed as:

$$\mathbf{Z} = (\mathbf{X} \circ \mathbf{W}) \star \mathbf{Y} \tag{1}$$

where $\circ$ corresponds to $f1()$ and $\star$ corresponds to $f2()$. Table 1 shows some examples of GEMM-Ops, divided into two groups. Group 1 includes all the GEMM-Ops where the $\circ$ operator can be of the $+/\times$ kind while $\star$ can be min/max. Group 2 contains the GEMM-Ops kernels where the $\circ$ operator also belongs to the min/max kind. $\mathbf{X}$ is a matrix of size $M \times N$, $\mathbf{W}$ is a matrix of size $N \times K$, while $\mathbf{Z}$ and $\mathbf{Y}$ have size $M \times K$.

The similarity of GEMMs and GEMM-Ops makes matrix computing units good candidates to be extended for supporting GEMM-Ops, extending their flexibility to accelerate generalized parallel algebraic operators. This class of algorithms

is also well-suited for ML applications since matrices are the baseline structure of all DL models. To this purpose, it is essential to note that the structure of Equation 1 is symmetric. As a consequence, for ML applications, there is no need to identify **X** or **W** as input or weight matrices because their role can be flexibly exchanged.

3.2. Asymptotic Optimality of Linear Algebra Acceleration Strategies

Memory load/store operations enlarge the gap between theoretical and practical performance and efficiency. Therefore, maximizing the number of operations performed per memory access, i. e. the arithmetic intensity, is the key to an efficient design. As analyzed by Pedram [47], scalar dot products and vector units do not guarantee the best trade-off between the number of operations performed per memory load/store access. As shown in Fig. 1a, a simple scalar dot product that operates on a N-dimensional array performs $2 \times N$ operations (N multiplications + N additions). The memory operations performed in this kernel are N loads of X, N loads of W, one load of Y and one store of Z. The resultant arithmetic intensity is:

$$Intensity\_1D = \frac{OPs}{LD/ST} = \frac{2N}{2N+2} \sim 1, \quad (N \to \infty). \qquad (2)$$

2-Dimensional $L \times H$ arrays exploit block-dot products (outer product) microkernels to perform GEMMs. Let us consider an $L \times H$ 2D array that can operate on $L \times 1$ and $1 \times H$ vectors, each made of N elements, like those shown in Fig. 1b. The operations performed on the two vectors are $2 \times L \times H$, repeated N times. The resulting load/store operations are $L \times N$ loads of X, $H \times N$ loads of W, $L \times H$ loads of Y and $L \times H$ stores of Z. With these changes, Equation 2 becomes:

$$\frac{OPs}{LD/ST} = \frac{2LHN}{(L+H)N + 2LH} \sim \frac{2LH}{L+H}, \quad (N \to \infty). \qquad (3)$$

Equation 3 shows that if $L = H$, the number of operations is quadratic with the size of the 2-D array, while the number of memory accesses remains linear. This demonstrates that 2-dimensional arrays are more efficient with respect to scalar or vector units. Thus, we will exploit the outer-product approach for the RedMulE design.

4. Architecture

In this section, we describe the PULP cluster, the hardware template we rely upon, and the RedMulE micro-architecture.

4.1. PULP Cluster and RedMulE

In Fig. 2, we show the architecture of a PULP cluster, a multi-core architecture that features a parametric number (2–16) of 32-bit RISC-V general-purpose cores featuring a partially shared, partially private instruction cache. In this specific work, we focus on a PULP cluster containing 8 RISC-V cores, equipped with 128 kB of Tightly-Coupled Data Memory (TCDM) split among 16 banks for word-level interleaving with

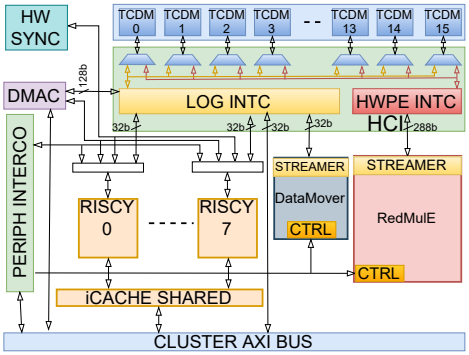


Figure 2: PULP cluster architecture with HWPEs integration.

a low level of contention. The PULP cluster also features an event unit for flexible internal synchronization and a dedicated Direct Memory Access Controller (DMAC) to efficiently move data between the TCDM and external memories. A peripheral interconnect allows the RISC-V cores to program the on-board peripherals (like the DMAC), and an AXI4 full cross-bar interconnect allows communications with the external environment. The PULP cluster also features a system-level clock gating cell to eliminate its dynamic power consumption when the cluster is not in use.

The capabilities of the PULP cluster can be further enhanced by integrating application-specific hardware accelerators called Hardware Processing Engines (HWPEs). HWPEs are software programmed by the RISC-V cores through the peripheral interconnect and share the TCDM with the RISC-V cores and the DMAC. In this sense, the HWPEs are tightly-coupled with the cluster cores [48]. The cores, the DMAC, and the accelerators access the shared TCDM through a single-cycle latency Heterogeneous Cluster Interconnect (HCI) [49]. Such interconnect features a *logarithmic* branch that allows all-to-all single-cycle accesses from 32-bit master ports, like those of the cores or the DMAC, to each of the word-interleaved memory banks. Conflicts are managed by granting only one initiator per bank with a round-robin scheme. The other branch is the *shallow* branch. It features a single n-bit parametric port, routed to adjacent 32-bit memory banks treated like a single wider bank without arbitration. This branch allows for simple integration of tightly-coupled accelerators like HWPEs. The bitwidth of the shallow branch port can be tuned to the HWPE requirements through a parameter. The TCDM banks are connected to the two HCI branches through a set of multiplexers, which grant access to one branch or the other according to a configurable starvation-free rotation scheme, allocating a configurable maximum of K < N consecutive cycles to the HWPE over a period of N cycles.

During the execution of NNs workloads, particularly during inference and training, on-the-fly data marshalling operations are known to reduce performance significantly. For this reason, our PULP cluster features a DataMover [49]. The DataMover is a tiny accelerator capable of transposing 3-dimensional tensors stored in the TCDM, with 33% less time than eight RISC-V cores and up to 50× increased energy efficiency (the lower the precision of chunks to transpose the more significant the advantages). The accelerator works with a configurable data element

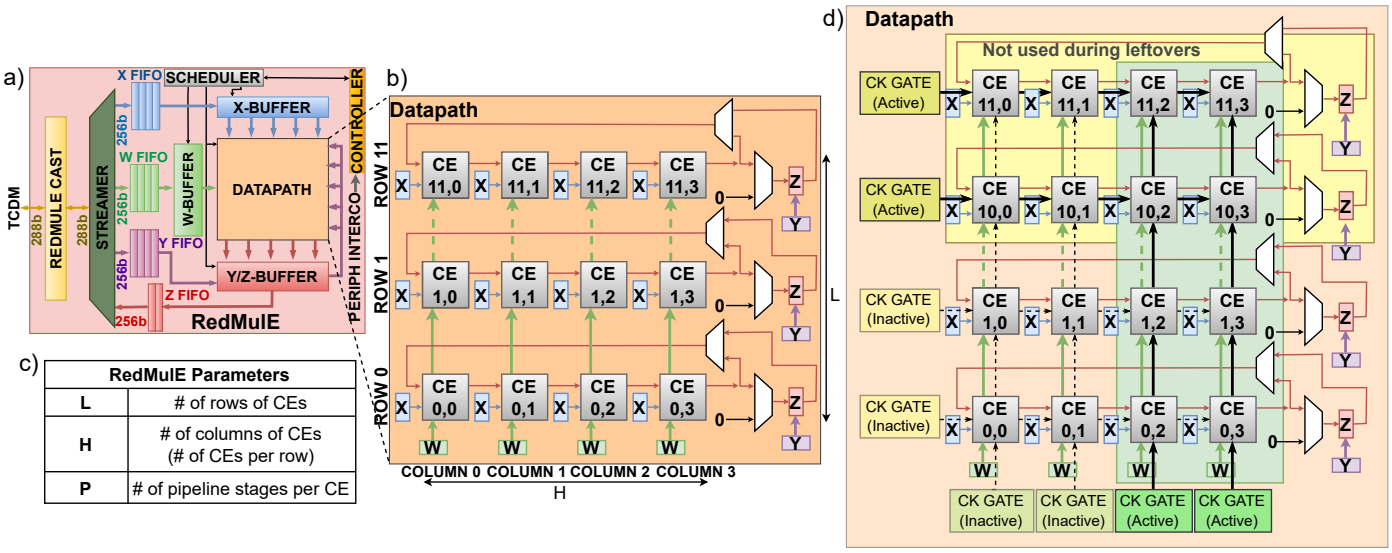


Figure 3: a) RedMulE internal architecture; b) RedMulE *Datapath* microarchitecture; c) Table with RedMulE design-time available parameters; d) Datapath-level clock gating logic.

bitwidth, from 32-bit down to 1-bit.

4.2. RedMulE

4.2.1. Global Architecture

RedMulE is a domain-specific processor designed to accelerate GEMM-Ops. Its architecture is shown in Fig. 3a. The core of RedMulE is the *Datapath*, a 2-Dimensional array of CEs interconnected as shown in Fig. 3b. The CEs are organized in L rows, each made of H columns. Within each row, a number of H CEs are cascaded so that each CE computing an intermediate product will pass its result to the next CE. The partial product computed by each row's last CE is fed back as accumulation input of the same row's first CE. The RedMulE *Datapath* features a design-time configurable number of internal CEs, pipeline registers (P) for each CE, and internal computing precision (FP bitwidth). All RedMulE's parameters are tunable at design time and are resumed in Fig. 3c.

Fig. 3d shows part of the hierarchical clock gating logic implemented within RedMulE. The proposed power-saving scheme leverages a fine-grained clock-gating mechanism that reduces the active power consumption of unused portions (rows and columns) of RedMulE's *Datapath*. RedMulE can dynamically disable the rows of CEs that would remain unused during the calculations involving the leftovers. The same holds for the columns of CEs, where RedMulE gates the clock of a given column within the *Datapath* depending on the calculation phase. Within the *Datapath*, each CE is divided into two stages, each dedicated to selecting the $\circ$ and $\star$ operations shown in Table 1. Each CE relies on computing FP modules adapted from the open-source *FPnew* trans-precision Floating-Point Unit (FPU) [50] so that their internal pipeline registers could support backpressure coming from memory stalls during RedMulE's operation. Fig. 4 and Section 4.2.2 will describe in detail how the proposed clock gating scheme hierarchically propagates within the CE's micro-architecture, where dedicated clock gating cells are in charge of disabling the unused units.

To feed the *Datapath* with data, RedMulE includes the *Streamer*, following the HWPE design strategy [1]. The *Streamer* is a specialized memory access unit that connects RedMulE to the HCI shallow branch through a single wide port of parametric size (multiple of 32-bit), used for load and store operations. The incoming stream from the HCI is propagated to a single input-multiple output dispatcher that forwards the valid only to the selected output channel; simultaneously, each output channel propagates the incoming stream from the HCI to the accelerator input ports. On the other hand, the streams produced by RedMulE are propagated to the HCI interface during write operations.

The *Streamer* is connected to three internal buffers: an X-Buffer that changes all the L inputs of a column once every $H \times (P + 1)$ cycles; a W-Buffer made of H shift registers, each broadcasting a new **W**-element to all the L CEs of a column every cycle; a Z-Buffer that buffers the computed **Z**-elements. The same buffer is used to pre-load **Y**-elements and push them into the *Datapath*. This solution saves area and power in the accelerator since there is no need for a separated buffer to store **Y** bias. All three buffers benefit from the clock gating logic shown in Fig. 3d. In particular, during the calculations of leftovers in input and output matrices, some of the rows of the *Datapath* do not perform any computation, and the corresponding lines in the X and Z buffers are clock gated as well. The same consideration holds for the columns of CEs and the lines in the W buffer. If a column of CEs does not perform any computation due to leftovers' presence, the corresponding line in the W-buffer is dynamically clock gated.

The control side of the accelerator is divided into two submodules, namely *Scheduler* and the *Controller*, that contain the register file, accessed by the cores to program the accelerator and cooperate to regulate the accelerator execution.

[1] https://hwpe-doc.rtfd.io

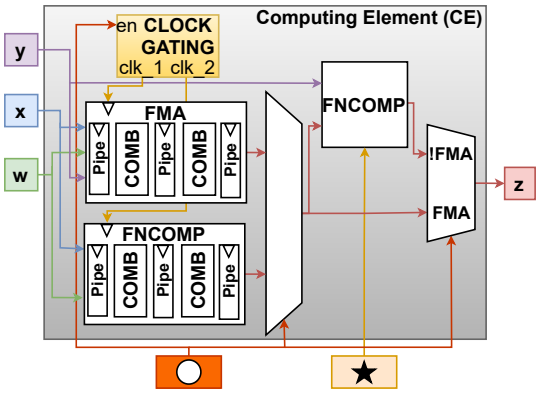


Figure 4: Microarchitecture of RedMulE's CE with extensions for GEMM-Ops support

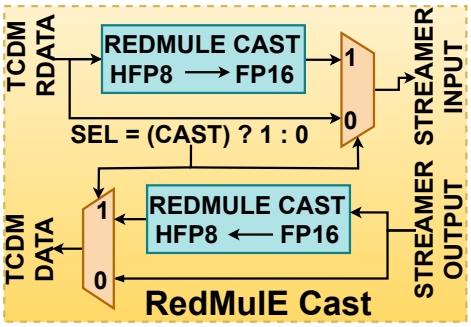


Figure 5: Internal architecture of RedMulE's cast module for mixed precision support.

4.2.2. GEMM-Ops Extension to the Computing Element

We extended each CE of RedMulE with dedicated hardware to support the execution of all the GEMM-Ops in Table 1. Fig. 4 shows the microarchitecture of each CE, which features two stages. The first stage selects the $\circ$ operation and contains one Fused Multiply-Add (FMA) unit and one Floating-Point Non-Computational Operations (FNCOMP) unit which implements FP MIN/MAX operations. $\mathbf{X}$ and $\mathbf{W}$ elements are then propagated to both the FMA and FNCOMP modules. Depending on the desired $\circ$ operation, a multiplexer selects the result of either the FMA or the FNCOMP. At the same time, the clock gating module shown in Fig. 4 freezes the input operands of the unused modules to completely eliminate undesired switching activity and related dynamic power. The FMA and FNCOMP feature P pipeline registers each, guaranteeing that the two modules introduce the same latency. We adapted the pipeline of both FMA and FNCOMP modules from the open-source *FPNew* transprecision FPU [50] to make them capable of supporting back pressure coming from memory stalls during RedMulE's operation.

In each CE, the $\mathbf{Y}$ element is propagated to the FMA unit in the first stage and is also directly connected to the input of the second stage of the CE. The second stage of each CE allows the $\star$ operation selection and contains a fully combinational FNCOMP module. The output multiplexer allows choosing the output of the second stage FNCOMP, in case of GEMM-Op, or that of the first stage FMA if a simple GEMM operation is performed. The proposed architectural solution guarantees the execution of all the operations listed in Table 1 with a compact implementation, in which we add just what is strictly needed.

4.2.3. Mixed-Precision Extension

Hybrid FP8 precision formats can be used as an efficient compression scheme to enable DL inference and training on extreme-edge devices. Hybrid FP8 precision means that the $\{sign, exponent, mantissa\}$ structure used to represent the tensors can be either $\{1, 5, 2\}$ or $\{1, 4, 3\}$[2]. The former format is best suited for backward propagation of gradients, as it provides a larger dynamic range but a lower accuracy. At the same time, the latter is a better fit for forward propagation of activations thanks to the larger mantissa [11, 10]. While 8-bit representation effectively reduces data footprint, it could severely impact accuracy due to reduced precision accumulations. To support this use case, RedMulE works internally with fixed FP16 precision but still accepts compressed FP8 formats as inputs and can generate FP8 compressed output tensors. To do this, we extended RedMulE's architecture with a dedicated casting module placed between the Streamer and the HCI interface, as shown in Fig. 3a.

Fig. 5 shows the architectural implementation of the casting module. It contains two FP cast units: the input one casts 8-bit FP incoming streams into 16-bit FP to feed the accelerator so that the CEs in the Datapath can operate on larger precision, guaranteeing high accuracy during intermediate accumulations. After the computation, the output cast unit converts the 16-bit FP results produced by RedMulE to 16-bit or 8-bit encoded outgoing streams before writing it to memory. The cast units can be excluded from the path if the input tensors are represented with 16 bits.

For DL use cases only, RedMulE can also be instantiated at design time to only load and store HFP8 operands. In this use-case, the input and output tensors represented with 8-bit formats allow to read or write from and to the memory twice the number of elements while keeping the same memory bandwidth. Consequently, this allows for doubling the number of CEs inside each row, doubling RedMulE's performance compared to the 16-bit inputs case.

4.3. RedMulE Computational Model

Fig. 6a shows how RedMulE performs a GEMM-Op visualising it on the computed matrices, while Fig. 6b and Fig. 6d show the detailed sequence of the operations within a row of CEs providing an example of GEMM execution. For this discussion, let us focus on a RedMulE implementation that features $L = 12$, $H = 4$, and $P = 3$. The RedMulE operation starts by pre-loading the Z-Buffer with L rows from the $\mathbf{Y}$-matrix, each row made of $H \times (P + 1) = 16$ FP16 elements (256-bit memory width/16-bit internal precision), namely $\mathbf{y_{0,0}}$ - $\mathbf{y_{0,15}}$ for Row_0, $\mathbf{y_{1,0}}$ - $\mathbf{y_{1,15}}$ for Row_1, and so on. Afterwards, RedMulE pre-loads the X-Buffer as well, following the same pattern, and

[2] Also called E4M3 and E5M2 by NVIDIA [https://tinyurl.com/mkhbxj3v]

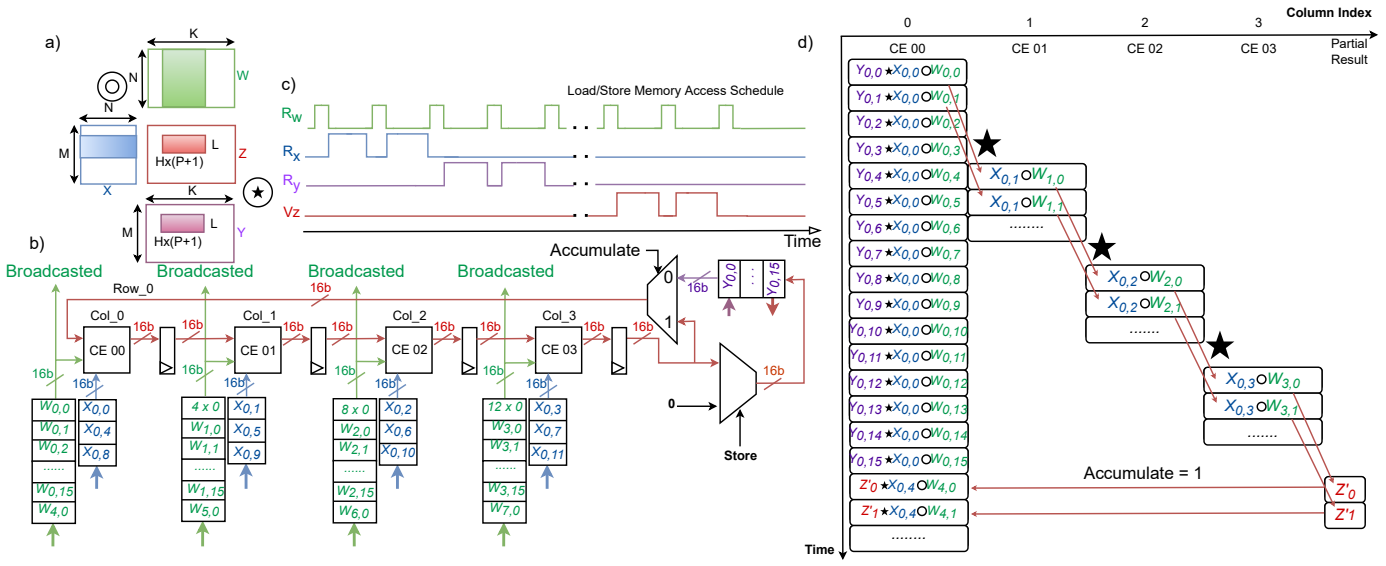


Figure 6: a) GEMM-Op execution displayed on matrices; b) Row of CEs within RedMulE *Datapath*; c) Memory access schedule in load/store mode described in terms of *R* (Ready) and *V* (Valid) handshake signals; d) Pipeline evolution within a row of ces.

then loads a set of $H \times (P+1) = 16$ **W**-elements ($\mathbf{w_{0,0}}$ - $\mathbf{w_{0,15}}$) inside the first shift register of the W-buffer. Each **W**-element is broadcasted to all the L CEs in the first *Datpath* column. While **W**-elements are broadcasted, the Z-Buffer pushes **Y**-elements in the CEs array cycle-by-cycle to perform the $\star$ operation during the execution of the $\circ$ one.

After $P+1$ cycles, each of the L CEs in the first column forwards its computed partial result to the neighbour CE in the second column. The accelerator loads another set of $H \times (P+1)$ **W**-elements ($\mathbf{w_{1,0}}$ - $\mathbf{w_{1,15}}$) to broadcast them to all the CEs in the second column. Once all the H CEs of a row have completed their computations, calculating a subset of $H \times (P+1)$ row-column intermediate results, RedMulE activates its feedback (*accumulate* = 1) to provide the intermediate results to the accumulation input of the first CEs of the given row, then reiterating the computation. Immediately after, the *Streamer* reloads the next **Y**-submatrix in the Z-Buffer so that it will be ready for the next calculation. During the Z-Buffer reload operation, the X-Buffer provides a new **X**-operand to the first column of CEs, and a new set of $H \times (P+1)$ **W**-elements is reloaded in the first W shift register. After $(P+1)$ cycles, all the L CEs of the first column produce a new partial product and provide it to the CEs in the second column. The X-Buffer provides a new **X**-operand at the input of the second column of CEs, and the W-Buffer loads a new set of $H \times (P+1)$ **W**-elements in the second W shift register for broadcasting, and the computation continues. Fig 6d shows the detailed sequence of data within the pipeline of a row of CEs from the beginning of a GEMM operation until the moment of the reuse of the partial results (*accumulate* = 1).

To guarantee a continuous data flow in the accelerator, the W-buffer accesses the memory once every $(P+1)$-cycles to load a new set of $H \times (P+1)$ **W**-elements. Once the X-Buffer and the Z-Buffer are empty, RedMulE reuses the *Streamer* port to load the **X** and **Y**-operands. Such operation is made by interleaving the memory accesses to **X** or **Y** matrices between two adjacent **W**-matrix accesses until the complete fulfilment of the X and Z buffers. Fig 6c shows how the memory accesses to different matrices are interleaved, describing the memory accesses in terms of Ready (R) and Valid (V) handshake signals. The *Streamer* load and store units fully support backpressure through a mechanism based on R/V handshake signals. Such a mechanism fully decouples the memory access and data consumption/production from the *Datapath*. The V signals for loads and the R signals for stores are generated within the *Streamer* itself depending only on memory stalls, which can be amortized by the presence of FIFO elements, and not on the actual usage from the *Datapath*. On the other hand, the *Datapath* uses the R signal of loads and the V signal of stores, as shown in Fig. 6c, to control the order of memory accesses interleaving them so that a continuous dataflow can be maintained. This choice is made to maximize the memory port utilization since having a single memory port also helps reduce the overall streamer area.

After the conclusion of an entire row-column operation, the Z-Buffer buffers the final sub-matrices. Afterwards, store operations are interleaved between two adjacent **W** load accesses until the Z-Buffer is empty and can be reloaded with **Y**-elements. With this approach, RedMulE optimizes the bandwidth utilization using a single wide memory port and achieves up to 99.4% CEs utilization.

5. Implementation and Measurements

5.1. Experimental Setup

We focus our experiments on a RedMulE$_{12x4}$ instance with $H = 4$, $L = 12$, $P = 3$, resulting in 48 CEs and a 288-bit wide HCI port, for 256-bit + 32-bit non-word-aligned accesses. We also address a RedMulE$_{12x8}$ since, as described in Section 5.2.3, it uses the same memory interface with twice the number of CEs.

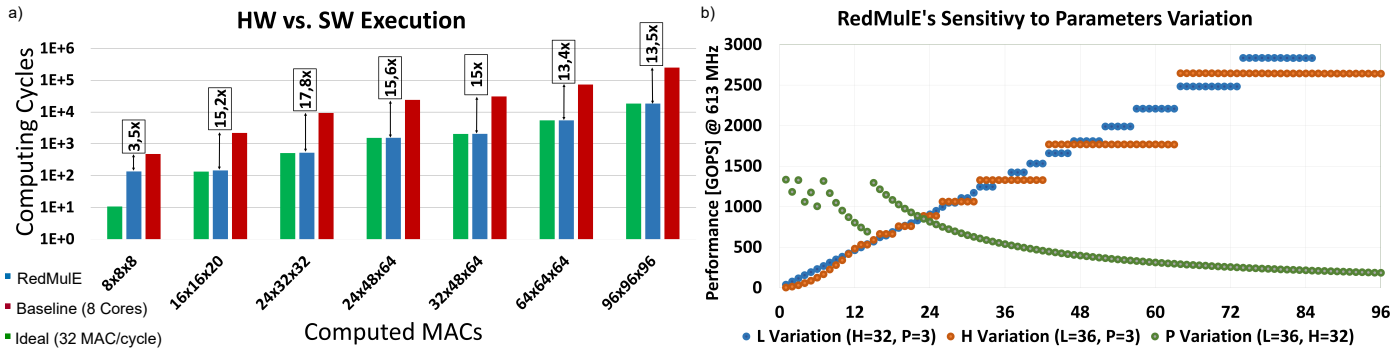


Figure 7: a) RedMulE's synthetic GEMM execution compared with software executed on 8 RISC-V cores; b) RedMulE's sensitivity to L, H, and P parameters.

Our experiments target GlobalFoundries 22 nm technology using Synopsys Design Compiler for synthesis (slow corner at $f_{\text{targ}} = 250\,\text{MHz}$, $V_{DD} = 0.59\,\text{V}$, $T = 125\,^{\circ}\text{C}$) and Cadence Innovus for full-cluster Place&Route in the same operating point. RedMulE's timing analysis and power extraction were made using Prime Time with 100% annotated switching activity from post-layout simulation in typical corner at 25 °C, targeting two operating points: 470 MHz at 0.65 V for high energy efficiency and 613 MHz at 0.8 V for high performance.

5.2. Performance Evaluation

5.2.1. GEMM Performance Evaluation

We use square and rectangular matrices as a synthetic benchmark to evaluate RedMulE's computation latency in cycles against the SW execution on 8 parallel RISC-V cores sharing 4 FPUs. On the given benchmark, RedMulE reaches a peak throughput of more than 95.4 OP/cycle, where we count both $\star$ and $\circ$ as one "OP", e.g. for a regular GEMM we count 1 MAC = 2 OPs. RedMulE achieves up to 99.4% of CEs utilization on 96×96 FP16 matrices (55 kB memory occupation), leading to 58.5 GFLOPS at 613 MHz with 0.80 V supply. Fig. 7a shows the number of computing cycles required to compute various matrices during parallel FP16 software executed on 8 RISC-V cores and compares them on RedMulE, showing that it reaches 15× average speedup over the software on large matrices. This performance increase with respect to the software counterpart settles around 13× with larger matrices since also the software execution becomes more efficient in those cases. We also consider the acceleration of a small $8 \times 8 \times 8$ case, as shown in Fig. 7a in which the accelerator is under-utilized, but it still introduces 3.5× speedup over the software parallel execution.

We evaluated the sensitivity of RedMulE to the L, H, and P parameters variation by executing a GEMM kernel with dimensions fixed to M=512, N=512, and K=512. We show the effect of parameters' variation in Fig. 7b. Each RedMulE tile features L rows and $H \times (P+1)$ columns. For this reason, each computation of RedMulE accesses the $\mathbf{X} - \textit{matrix}$ a number of times equal to $\frac{M}{L} \times \frac{N}{H\times(P+1)}$, while it accesses the $\mathbf{W}$-matrix a number of times equal to $N \times \frac{K}{H\times(P+1)}$. Looking at the accelerator behaviour when sweeping the L parameter (the number of rows within the RedMulE datapath), considering that each tile of the matrix depends on the ratio $\frac{M}{L}$ (number or rows of the $\mathbf{X}$-matrix divided by the number of rows in the accelerator), the performance of RedMulE rises linearly until the tile ratio starts to flatten. When this happens, the performance behaviour starts stepping up every time an increase of the L value allows to reduce the number of iterations on the $\mathbf{X}$-matrix. Then it flattens again until the next step happens.

Similar considerations hold for the sweep of the H parameter. Since RedMulE's accesses to both $\mathbf{X}$-matrix and $\mathbf{W}$-matrix depend on $H \times (P+1)$, as H increases, the performance of RedMulE increases until the ratio $\frac{N}{H\times(P+1)}$ and $\frac{K}{H\times(P+1)}$ flattens. Then, the performance curve steps up every time an increase of the H parameter reduces the number of accesses to the $\mathbf{X}$-matrix and the $\mathbf{W}$-matrix before flattening again. Since the value of L is upper-bounded by $H \times P$, the value of L also changes in the first part of the sweep accordingly with the $H \times P$, explaining the initial non-linear behaviour of the curve.

Compared with the previous cases, the performance trend changes when sweeping the P. Increasing the number of pipeline registers not only traduces into larger internal tiles and thus reduces the number of times RedMulE accesses the $\mathbf{X}$ and $\mathbf{W}$ matrices, but also turns into a higher internal capacity of the accelerator. By keeping the input matrices' dimensions fixed, RedMulE's performance drops as pipeline registers increase because progressively fewer data are available to feed the internal pipeline queues. Consequently, the leftovers and the computation time increase, while the performance drops due to higher latency introduced by the pipeline registers. When the value of P creates a tile that reduces the number of times RedMulE accesses the matrices the performance increase again, then drops with parabolic behaviour because the performance is inversely proportional to P.

5.2.2. FP16 Network Training

To further evaluate RedMulE performance on a real-case NN training, our target is TinyMLPerf [51], and in particular, we focused on the ResNet [52] example. For the software infrastructure, we rely on the pulp-TrainLib [26], and we compared RedMulE with a software baseline executed on 8 RISC-V cores sharing 4 FPUs. The library takes into consideration all the training steps for the calculation of the gradients and backpropagation. Fig. 8a shows the execution of a single step in the ResNet8 network when using 8 RISC-V cores

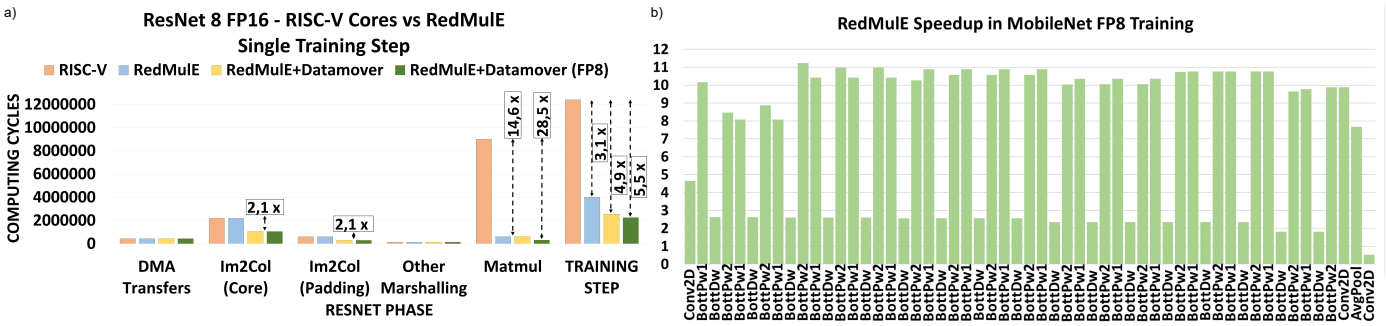


Figure 8: RedMulE performance on neural networks compared with software executed on 8 RISC-V cores: a) ResNet8 and b) MobileNet V2 training execution.

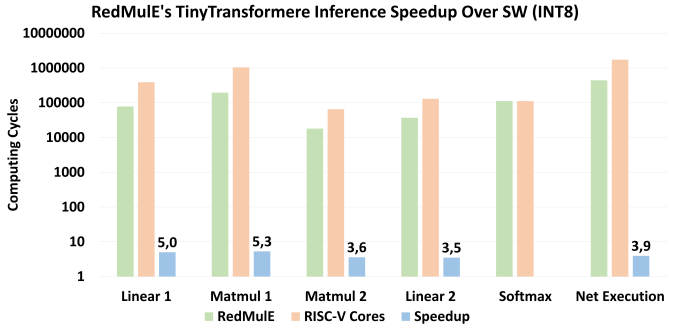


Figure 9: RedMulE performance on Tiny Transformer FP8 inference compared with parallel INT8 software executed on 8 RISC-V cores.

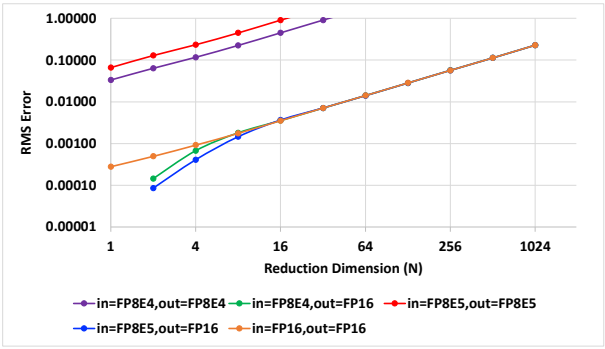


Figure 10: RMS Error analysis for different input/output floating point formats over variable reduction N sizes.

in parallel and when using RedMulE for the matrix multiplication execution. RedMulE accelerates the matrix multiplication execution of 14.6× with respect to the parallel RISC-V execution in SW, speeding up the entire single step of the ResNet8 of 3.1×. RedMulE keeps its utilization constant at 99.1% (47.6 MAC/cycle) with the only exceptions in the first and the last layers where it drops to 93.2% (44.7 MAC/cycle) and 32.3% (15.5 MAC/cycle) due to leftovers that do not allow to exploit the full potential of the array. From Fig. 8a, it is also evident that the data reorganization during the Im2Col accounts for approximately 3 Millions computing cycles. To solve this problem, we augment RedMulE's operation with the support of the DataMover engine, halving the number of computing cycles required to perform the two Im2Col operations and thus speeding up the overall training step execution up to 4.9×. As all the devices included in the PULP cluster (RISC-V cores and accelerators) are designed for synergistic cooperation and share the memory, the heterogeneity of the architecture can be efficiently and fully exploited.

5.2.3. *HFP8 Network Training*

For the same training example, we consider a RedMulE$_{12x8}$ instance used to train the ResNet network encoded on 8-bit FP inputs only. For the RedMulE$_{12x4}$ we considered until now, the memory port of the *Streamer* is 288-bit wide, meaning a 256-bit memory port with non-word aligned memory accesses capability. In this configuration, RedMulE$_{12x4}$ can load $16 \times FP16$ elements at a time that are used to fill the pipeline during the computation. Having $H = 4$ columns, the pipeline stages within each row are calculated as $H\times(P+1)$, where $P = 3$ in this implementation, resulting in 16 pipeline stages. Considering a fixed 8-bit input encoding, with the same 288-bit memory port, RedMulE can access up to $32 \times FP8$ elements at a time, meaning that we can implement a RedMule$_{12x8}$ instance maintaining a 288-bit memory interface and obtaining 32 pipeline stages. We show how the ResNet8 training can benefit from this configuration in the green bar of Fig. 8a. Matrix multiplication execution can be accelerated up to 28.5×, resulting in 5.5× speed-up over the entire training step execution, with 97% utilization.

In addition to the ResNet example, Fig. 8b shows our evaluation of the speedup introduced by RedMulE over parallel software during the training step of a MobileNet V2 [53] using FP8 format. RedMulE introduces a 7.5× average speedup with an 11.2× peak speedup over a carefully optimized software execution on 8 RISC-V cores with private FPUs. Interestingly, RedMulE accelerates the execution of depthwise layers by up to 2.6× despite the unfavourable sizes of the reshaped matrices that result in high underutilization of the accelerator. RedMulE accelerates the whole training step by 6.4× compared to the parallel execution on general-purpose cores.

5.2.4. *FP8 Transformer Inference*

We evaluated the performance of RedMulE during the execution of a TinyTransformer [54] network inference using FP8 since it is more suited for transformers as a data format [55]. Fig. 9 shows the results of the inference of the network, high-

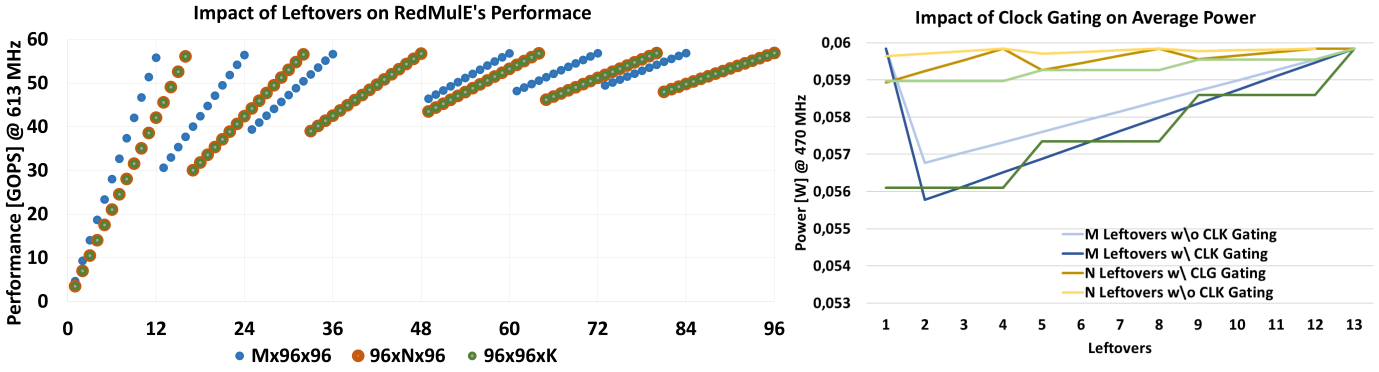


Figure 11: Impact of leftovers on RedMulE performance and energy efficiency with fine-grained clock gating.

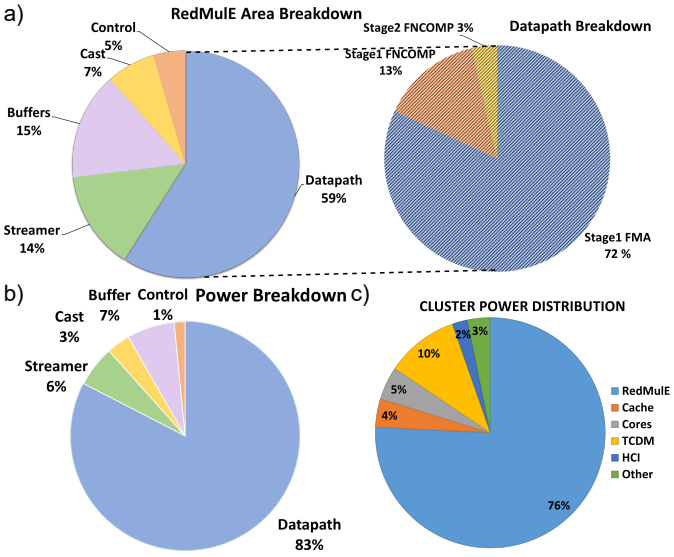


Figure 12: a) RedMulE area breakdown with a focus on the datapath, b) RedMulE power distribution, c) PULP power distribution.

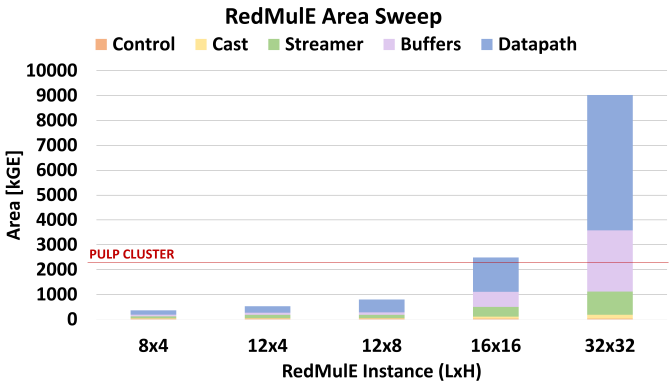


Figure 13: RedMulE area sweep with several sizes of H and L.

lighting that the execution of both linear and matmul layers significantly benefits the presence of RedMulE when compared with parallel execution on the general-purpose processors of the cluster. Despite the execution on the general-purpose cores benefits of SIMD extensions with 8-bit integer arithmetic, RedMulE introduces more than 4× average speedup on all the network layers, with a peak 5.3× speedup on the Matmul1 layer and a 3.9× speedup on the execution of the entire network.

5.3. Error Analysis on Input and Output Tensors Format

We performed error analysis considering an increasing value of the **X** and **W** matrices' reduction dimension (N). The FMA unit in the CE works with fixed internal precision, reducing precision loss during the intermediate accumulation steps. However, when **X**, **W**, and **Z** are all represented with 8-bit precision, the RMSE increases more than 100× compared to the 16-bit into 16-bit case, with a trend exponentially dependent on the N size. However, by representing only **X** and **W** with 8-bit precision and keeping a wider 16-bit precision for the output **Z**-matrix, the accuracy loss is negligible compared to the 16-bit only case.

5.4. RedMulE Area

5.4.1. Area Breakdown analysis

RedMulE_{12x4} occupies 0.15 mm^2, corresponding to 23.8% of the entire PULP cluster area (0.64 mm^2). Fig. 12a shows the breakdown of the RedMulE area, where the cast units account for the 7% to the overall accelerator area, and the CEs account for the 59%, the 72% of which is given by the FMA units.

5.4.2. RedMulE Area Sweep

We studied the area overhead introduced when changing the number of CEs within RedMulE, fixing the CEs' internal pipeline stages to $P = 3$. Fig. 13 shows that RedMulE's area occupation becomes comparable to the area of the entire PULP cluster when it contains 256 CEs, corresponding to a RedMulE_{16x16} instance. On the other hand, the area of RedMulE_{32x32} is 4× larger than the entire PULP cluster. Fig. 13 shows that changing the shape of the *Datapath* also affects the size of the *Streamer*. In particular, for each CE that is added to a row of the *Datapath* (or equivalently, changing the H parameter), other $P + 1$ pipeline registers are added within each *Datapath* row. The consequence is that the number of elements needed to keep a high CEs utilization increases by $P+1$ as well. Keeping $P = 3$ as an example, increasing the H parameter by 1 requires the *Streamer* to provide $P + 1 (= 4)$ additional FP16 elements to the *Datapath*. The consequence is that the streamer port must be enlarged of 64-bit ($= 4 \times 16 - bit$), limiting the integration of RedMulE in the PULP cluster.

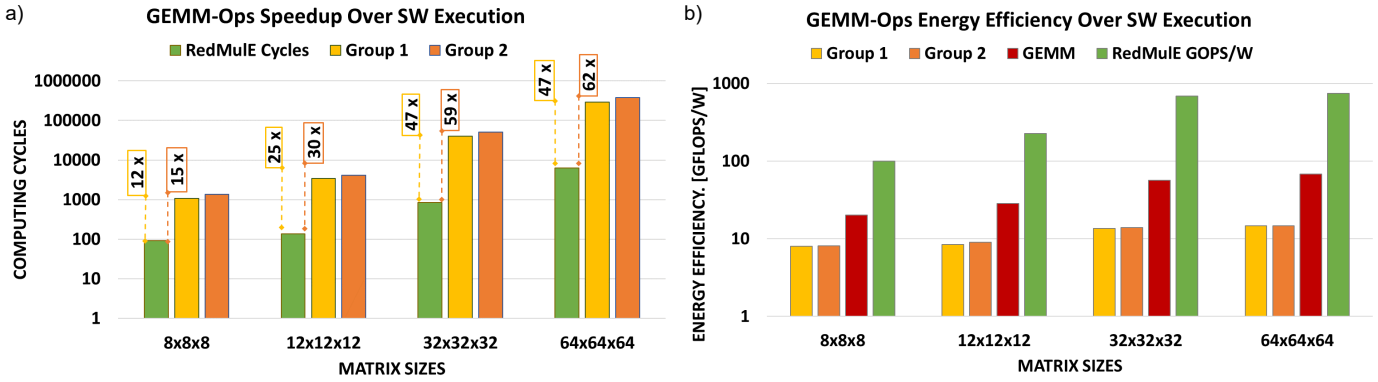


Figure 14: a) Performance and b) energy efficiency of RedMulE during GEMM-Ops execution over the parallel software counterpart.

5.5. RedMulE Power

At a cluster level, the power consumption in the efficiency point amounts to 59.3 mW during GEMM operation. The RedMulE contribution dominates the power envelope accounting for 66.8% of the overall consumption, while the TCDM banks and the HCI interconnect contribution is 13.3%. In this operating point, we reach a cluster peak energy efficiency of 755 GFLOPS/W during GEMM execution, corresponding to 12.5× higher energy efficiency with respect to the software baseline. Fig. 12b and Fig. 12c show respectively the power breakdown for RedMulE, where most of the power is consumed by the *Datapath*, and the PULP cluster during a GEMM operation, where the majority of the power is consumed by RedMulE and by the TCDM banks.

5.6. Clock Gating Evaluation

We evaluated how the effect of leftovers impacts the performance and energy efficiency of RedMulE and how fine-grained clock gating helps reduce power consumption during underutilization phases. Fig. 11 shows our analysis results: to evaluate performance loss, we vary one of the dimensions of the input matrices (M, N, or K) at a time and keep steady the other two, fixing RedMulE's parameters to L=12, H=4, and P=3. To evaluate the power effect, we analyze the power consumption of RedMulE with and without the clock gating contribution in a window corresponding to the last performance step.

In Fig. 11, the blue dots in the performance diagram highlight the performance trend when we vary the number of **X**-matrix rows (M). Having leftovers in the rows of the **X**-matrix means that RedMulE uses only a certain number of rows (from one to L) to perform the computation of that piece of the matrix. Therefore, an entire part of the RedMulE array does not perform any calculation. In addition, the activity of the X and Z buffers is also lower because RedMulE needs only part of those buffers. From a performance viewpoint, Fig. 11 shows that the performance rises from 4.7 GOPS to 55.8 GOPS when M goes from 1 to 12. In this case, we have one single tile of the **X**-matrix being multiplied by all **W**-matrix tiles. Thus the leftovers impact the entire matrix-matrix operation. By increasing M, the number of computing cycles with full array utilization gradually increases, thus balancing the performance losses caused by the presence of leftovers.

The right part of Fig. 11 shows the power trends resulting from clock gating the unused computing elements in the array and the unused portions of the X and Z buffers during the leftovers' calculation. The proposed clock gating scheme saves up to 22% of the accelerator's power when the number of **X**-matrix rows (M) is much lower than the number of **X** columns (N). In this case, a large portion of the CEs array is underutilized, and thus clock-gated during most of the computation. Such a situation occurs, for example, during the execution of bottleneck depthwise layers of the MobileNet, which can be reshaped into vector-matrix multiplications ($M = 1, N >> 1$).

The effect of varying N and K produces similar results. Leftovers on N and K turn into a low utilization of the pipeline registers of each accelerator row. From a performance viewpoint, this is similar to the case in which M changes. Still, the curves have a lower slope because, with H=4 and P=3, each row has 16 ($= H \times (P + 1)$) internal pipeline registers and every time the value of N or K reaches a multiple of $H \times (P + 1)$ each plot line approaches the maximum performance value. Similarly to the case with variable M, also in this case, the first curve starts from a minimum performance value of 3.5 GOPS because the leftovers impact the entire computation. From the second to the last lines, the computing cycles in which RedMulE fully uses the array of CEs compensate for the impact of matrix leftovers. From the power consumption perspective, the plot lines step once every four matrix elements because RedMulE features four ($= P + 1$) pipeline registers between each column of CEs. The proposed clock gating methodology allows gating columns of CEs that are not active during leftovers' computation so that there can be from one (the first step in the power consumption lines) to four (the last step in each power consumption line) CEs active when calculating leftovers. The proposed clock gating scheme saves up to 37% of average power in heavy underutilization conditions, thus increasing the power efficiency by 60% compared to the non-clock-gated case.

5.7. GEMM-Ops Measurements

To evaluate the GEMM-Ops performance, in Fig. 14a we compare the RedMulE GEMM-Ops execution against parallel SW execution on the RISC-V cores. RedMulE always takes the same number of computing cycles to perform any of the supported GEMM-Ops. On the contrary, the parallel execution on

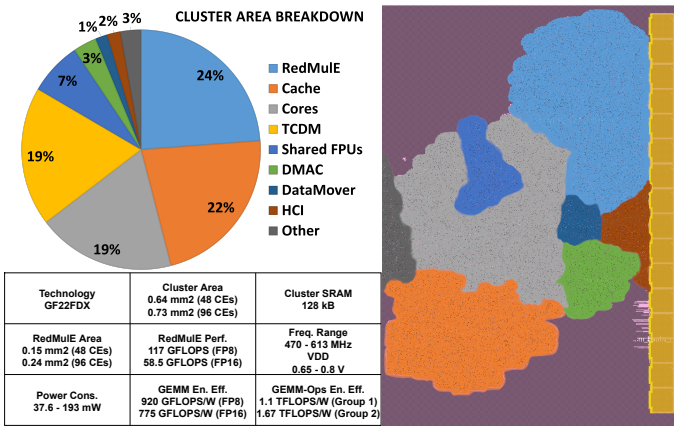


Figure 15: Area breakdown of the PULP cluster, layout and resume table.

the general-purpose cores changes depending on the executed kernel. All the kernels belonging to Group 1 (see Table 1), i.e. for which the $\circ$ operation is +/x and the $\star$ one is max/min, require the same number of computing cycles when executed on the cores. The execution of these kernels on RedMulE allows for up to 47× speedup. When also $\star$ are of the max/min kind, i.e. Group 2, the execution overhead for the general-purpose cores is even higher, and RedMulE can accelerate such kernels up to 62×.

Fig. 12a shows that the support for GEMM-Ops introduces an area overhead of just 16% over the entire accelerator area. 13% of this overhead resides in the first stage FNCOMP unit, dominated by the pipeline introduced to match the latency cycles of the FMA unit. The second stage FNCOMP unit is fully combinational and accounts only for 3% overhead.

We evaluated the power consumption and the energy efficiency of RedMulE during GEMM-Ops execution. Fig. 14b compares the energy efficiency of RedMulE with the software baseline executed on 8 RISC-V cores with 4 shared FPUs on FP16 elements during the execution of GEMM, GEMM-Ops' Group 1 and GEMM-Ops' Group 2 kernels. For the GEMM-Ops' Group 1, the cluster-level power dissipation reaches 53.2 mW, leading to 842 GFLOPS/W, 57.2× better than the parallel SW execution on general-purpose cores. On the other hand, during the execution of the algorithms in GEMM-Ops' Group 2, the power consumption is further reduced to 37.6 mW resulting in 1.19 TFLOPS/W, with an energy efficiency increase of 81.2× compared to the parallel software counterpart.

6. Comparison with the State-of-the-Art

Table 2 resumes the comparison of our work with different State of the Art (SoA) architectures.

We compare our work with GPU architectures, in particular with NVIDIA H100 containing TensorCores, that guarantee up to 989 TFLOPS of performance in FP16 and 1979 TFLOPS in FP8, meaning 17000× than our work, but at the cost of 700 W power consumption and 814 mm^2, 12000× more power-consuming and 1300× larger than our work – representing an unfeasible solution for an IoT end-node.

While RedMulE targets primarily training, it is also usable for inference. For this reason, we include in our comparison some inference-oriented chips, like DNPU [21]. DNPU's performance is just 1.9× higher than our cluster, although DNPU contains 16× the number of CEs. Moreover, DNPU features 2.7× higher efficiency than RedMulE but is designed to work with fixed-point precision only, which helps increase energy efficiency. We also compared our work with Diana [20] and Gemmini [22], being designed in the same technology node of RedMulE. The former achieves 44.5% less performance than RedMulE_{12x8} and 12% less performance than RedMulE_{12x4} in the energy efficient mode. Diana's power consumption in efficiency mode is much lower than our design, but if we scale down the frequency to 50 MHz as they do, our PULP cluster with RedMulE_{12x4} consumes just 7.65 mW. The significant increase in Diana's energy efficiency is justified as it uses only 8-bit integer arithmetic, which helps reduce power consumption and increases energy efficiency. On the contrary, Gemmini features one order of magnitude less energy efficiency than RedMulE_{12x4} despite it features 5× the number of CEs and works with 8-bit integer format.

We also compared our design with other platforms specifically designed for on-chip training. IBM [35] demonstrated a 4-core AI chip in 7 nm technology which is just 2.4× more energy-efficient, 33.2× larger, and 74× more power-consuming than our PULP cluster with RedMulE_{12x4}, despite the technology scaling and the reduced operating voltage. IBM also proposes a chip [37], with more than 1 W of power consumption, which is not acceptable for extreme-edge training. On the other hand, LNPU [38] is an extreme-edge processor that features a 6.67× higher power envelope than RedMulE_{12x4}. Vega is a valid candidate for on-chip embedded training, but RedMulE_{12x4} achieves 7.8× higher performance and 3.2× higher energy efficiency. Cambricon-Q [30] is designed in 45 nm and is 2.9× more energy-efficient than our design but makes use of narrow 8-bit fixed-point arithmetic, while generally available learning algorithms based on backpropagation strictly require FP range and precision. Cambricon-Q is also 17.7× more power-hungry than our design, therefore not suitable for TinyML applications. Similar considerations hold for T-PIM [39], a training chip designed in 28 nm technology that features an in-memory computing core for high energy efficiency but only works with 16-bit integer precision, not satisfying the precision requirements to enable on-chip training.

TSUNAMI [41] and Trainer [42] are conceived for energy-efficient embedded training and extensively use pruning and sparse matrices generation to increase energy efficiency and reduce the number of required MAC operations during training with zero-skipping. We compare with the results they provide during dense calculations. In their best efficiency points, TSUNAMI and Trainer's power consumption is comparable to RedMulE's. However, those points correspond to 50 MHz and 40 MHz for TSUNAMI and Trainer, while RedMule is evaluated at 470 MHz. Therefore, RedMulE would consume approximately one order of magnitude less power at comparable frequencies. TSUNAMI and Trainer reach up to 5× and 8× higher performance, respectively, since they feature 21× and 85× the

Table 2: State of the art comparison. First line = Best Efficiency; Second line = Peak Performance. 1 MAC = 2 OPs.

Category	**Design**	**Tech** *nm*	**Area** mm^2	**Freq** *MHz*	**Volt** *V*	**Power** *mW*	**Perf** *GOPS*	**Energy Eff** *GOPS/W*	**CEs**	**Precision**
GPU	NVIDIA H100 [15]	4	814	1830	-	700000	1978900 989400	2827 1413	528	FP8 FP16
	$SIMD^2$ [6]	45	19.5	-	-	4190	-	-	-	INT16
Inference Chips	DNPU [21]	65	16	50 200	0.7 1.1	34.6 279	72.6 279	2100 1000	768	INT16
	Diana [20]	22	8.91	50 280	0.55 0.9	9.96 129	40 224	4040 1740	256	INT8
	Gemmini [22]	22	16	700 900	0.75 0.91	- -	- -	70 50	256	INT8
Training Chips	4-core IBM [35]	7	19.6	1000 1600	0.55 0.75	4400 13000	8000 12800	1800 980	4096	FP16
	LNPU [38]	65	16	200	0.78 1.1	367	600 300	1630 817	768	FP8 FP16
	Oh, IBM [37]	14	9.8	1000 1500	0.54 0.62	1428 2727	2000 3000	1400 1100	128	FP32 FP16
	T-PIM [39]	28	5.04	50 280	0.75 1.05	5.25 51.2	39.8 43	7590 840	-	INT16
	TSUNAMI [41]	65	16	50 200	0.78 1.1	45 419	612	3420 1480	2048	FP8
							306	1710 740	1024	FP16
	Trainer [42]	28	21	40 440	0.56 1	23 363	900	4280	8192	FP8
							450	2140	4096	FP16
	Cambricon-Q [30]	45	888	1000	0.6	1030	2000	2240	1024	INT8
	Vega [29]	22	12	450	0.5 0.8	- 49.4	3.3 7.5	250 180	4	FP16
Mat-Mul	Anders [43]	14	0.024	2.1 1090	0.26 0.9	0.023 82.7	0.068 34	2970 420	16	FP16
GEMM	**This Work RedMulE$_{12x4}$**	**22**	**0.64**	**470** **613**	**0.65** **0.8**	**59.3** **116**	**44.8** **58.5**	**775** **506**	**48**	**FP16**
Group 1 GEMM-Ops						**53.2** **103**		**842** **576**		
Group 2 GEMM-Ops						**37.6** **71.5**		**1193** **819**		
GEMM	**This Work RedMulE$_{12x8}$**	**22**	**0.73**	**470** **613**	**0.65** **0.8**	**97.5** **193**	**89.7** **117**	**920** **608**	**96**	**FP8**
Group 1 GEMM-Ops						**85.2** **168**		**1052** **694**		
Group 2 GEMM-Ops						**54** **104**		**1666** **1123**		

number of CEs, but feature a much lower CEs' utilization than our RedMulE (75% TSUNAMI and only 12.5% Trainer). The systolic architecture of RedMulE enables, in principle, almost arbitrary architecture scaling. Assuming linear performance, area, and power ratio, scaling to 1024 or 4096 CEs (21× and 85× larger than RedMulE$_{12x4}$), our utilization would still be 99.4%, leading to higher overall performance (1.25 TFLOPS and 5 TFLOPS respectively) once accounting overheads.

We compare RedMulE$_{12x4}$ with Anders *et al.* [43], proposing a hardware accelerator for matrix multiplications in 14 nm technology that targets TinyML learning and inference applications. It reaches a peak energy efficiency of 2.97 TFLOPS/W in FP16 precision, 3.83× higher than RedMulE$_{12x4}$, but only when operating at near-threshold voltage (260 mV) and extremely reduced frequency (2.1 MHz). In that operating point, their design is 659× less performant than RedMulE. Anders' peak performance is obtained at 0.9 V and 1.09 GHz, leading to 420 GFLOPS/W and 99.4% MAC units utilization, similarly to RedMulE's. In 22 nm technology, at 613 MHz frequency and 0.8 V, RedMulE$_{12x4}$ reaches 58.5 GFLOPS, 1.72× better than Anders *et al.*, with a 20.5% higher energy efficiency of 506 GFLOPS/W on FP16 GEMM.

We also compared RedMulE with SIMD^2 [6], the only other design that features GEMM-Ops extensions. Even though SIMD^2 works only with integer arithmetic and cannot thus target on-chip training, it features 36.1× higher power consumption than RedMulE. In their case, the authors also claim that the area overhead to build GEMM-Ops extensions on top of NVIDIA Streaming Multiprocessor accounted for 69%, while in RedMulE, the area overhead introduced by GEMM-Ops extension is just 16%.

7. Conclusion

In this paper, we presented RedMulE - Reduced-Precision Matrix Multiplication Engine, a fully-parametric open-source cluster-coupled accelerator enabling TinyML training on ultra-low-power devices, i.e. near-sensor training on a few tens of mW of power budget. RedMulE is conceived for FP16 GEMM-Ops computation, and supports compressed FP8 inputs while also efficiently accelerating a wide range of operations that share the same structure of a GEMM. RedMulE allows the instantiation of a wide range of Floating-Point Units-based Computing Elements (CEs), internal buffers, and memory interface configurations. We integrated an instance of RedMulE, containing a 12×4 array of CEs into an ultra-low-power cluster containing 8 RISC-V cores, and implemented the resulting system in a 22 nm technology. RedMulE achieves 99.4% CEs utilization and an average 15× speedup during simple GEMM execution with respect to a parallel software baseline running on the eight cores. It occupies 0.15 mm^2 accounting for only 24% of the cluster area. During GEMM-Ops execution, the performance speedup introduced by RedMulE over the RISC-V cores reaches up to 62×. In its best performance point (at 613 MHz, 0.8 V), RedMulE achieves 506 GFLOPS/W @ 58.5 GFLOPS when executing FP GEMM kernels; while, in its best efficiency point (at 470 MHz, 0.65 V), it reaches 775 GFLOPS/W @ 44.8 GFLOPS. On a real example of NN training, RedMulE accelerates the matrix multiplication by up to 14.6× and 28.5× when the input tensors are represented with 16-bit and 8-bit respectively, accelerating the whole training step of ResNet8 by 4.9× and 5.2×.

Acknowledgments

This work was supported in part by Thales Alenia Space, The European PILOT (EuroHPC JU, g.a. 101034126), and NeuroSoC (Horizon EU g.a. 101070634). Supported in part by "TinyTrainer" project that receives funding from the Swiss National Science Foundation (g.a. 20791).

References

[1] M. Satyanarayanan, The emergence of edge computing, Computer (Volume: 50, Issue: 1, January 2017) (2017).

[2] J. Gilbert, S. Reinhardt, V. B. Shah, A Unified Framework for Numerical and Combinatorial Computing, Computing in Science & Engineering (Volume: 10, Issue: 2, March-April 2008) (2008).

[3] J. Gilbert, S. Reinhardt, V. B. Shah, High-Performance Graph Algorithms from Parallel Sparse Matrices, Applied Parallel Computing. State of the Art in Scientific Computing. PARA 2006 (2008).

[4] M. Mohri, Semiring frameworks and algorithms for shortest-distance problems, Journal of Automata, Languages and Combinatorics 7 (3) (2002) 321–350.

[5] S. G. Sedukhin, M. Paprzycki, Generalizing matrix multiplication for efficient computations on modern computers, in: R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Waśniewski (Eds.), Parallel Processing and Applied Mathematics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 225–234.

[6] Y. Zhang, P. Tsai, H. W. Tseng, SIMD2: a generalized matrix instruction set for accelerating tensor computation beyond GEMM, ISCA '22: Proceedings of the 49th Annual International Symposium on Computer Architecture (2022).

[7] L. Ravaglia, M. Rusci, A. Capotondi, F. Conti, Memory-Latency-Accuracy Trade-Offs for Continual Learning on a RISC-V Extreme-Edge Node, IEEE Workshop on Signal Processing Systems (SiPS) (2020).

[8] NVIDIA, Training with mixed precision - nvidia deep learning performance documentation, `https://tinyurl.com/2p944wfs` (2021).

[9] A. F. Rodriguez Perez, B. Ziv, E. M. Fomenko, E. Meiri, H. Shen, Lower Numerical Precision Deep Learning Inference and Training, `https://tinyurl.com/457ymjfk` (2018).

[10] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, N. Mellempudi, S. Oberman, M. Shoeybi, M. Siu, H. Wu, FP8 Formats for Deep Learning, arxiv preprint arXiv:2209.05433 (2022).

[11] X. Sun, J. Choi, C. Chen, N. Wang, S. Venkataramani, Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks, IBM T. J. Watson Research CenterYorktown Heights, NY 10598, USA (2019).

[12] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, L. Benini, A transprecision floating-point platform for ultra-low power computing, in: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018, pp. 1051–1056. `doi:10.23919/DATE.2018.8342167`.

[13] L. Bertaccini, G. Paulin, T. Fischer, S. Mach, L. Benini, Minifloat-nn and exsdotp: An isa extension and a modular open hardware unit for low-precision training on risc-v cores, in: 2022 IEEE 29th Symposium on Computer Arithmetic (ARITH), IEEE Computer Society, Los Alamitos, CA, USA, 2022, pp. 1–8. `doi:10.1109/ARITH54963.2022.00010`. URL `https://doi.ieeecomputersociety.org/10.1109/ARITH54963.2022.00010`

[14] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, J. Kepner, Ai and ml accelerator survey and trends, in: 2022 IEEE High Performance Extreme Computing Conference (HPEC), 2022, pp. 1–10. `doi:10.1109/HPEC55821.2022.9926331`.

[15] NVIDIA, Nvidia h100 tensor core gpu architecture, https://resources.nvidia.com/en-us-tensor-core (2022).

[16] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, J. Zhang, Edge intelligence: Paving the last mile of artificial intelligence with edge computing, Parallel Processing and Applied Mathematics (2019).

[17] A. N. Mazumder, J. Meng, H.-A. Rashid, U. Kallakuri, X. Zhang, J.-S. Seo, T. Mohsenin, A survey on the optimization of neural network accelerators for micro-ai on-device inference, IEEE Journal on Emerging and Selected Topics in Circuits and Systems 11 (4) (2021) 532–547. doi:10.1109/JETCAS.2021.3129415.

[18] Y. Jin, J. Cai, J. Xu, Y. Huan, Y. Yan, B. Huang, Y. Guo, L. Zheng, Z. Zou, Self-aware distributed deep learning framework for heterogeneous iot edge devices, Future Generation Computer Systems 125 (2021) 908–920. doi:https://doi.org/10.1016/j.future.2021.07.010.
URL https://www.sciencedirect.com/science/article/pii/S0167739X21002715

[19] C. Åleskog, H. Grahn, A. Borg, Recent developments in low-power ai accelerators: A survey, Algorithms 15 (11) (2022). doi:10.3390/a15110419.

[20] P. Houshmand, G. M. Sarda, V. Jain, K. Ueyoshi, I. A. Papistas, M. Shi, Q. Zheng, D. Bhattacharjee, A. Mallik, P. Debacker, D. Verkest, M. Verhelst, Diana: An end-to-end hybrid digital and analog neural network soc for the edge, IEEE Journal of Solid-State Circuits 58 (1) (2023) 203–215. doi:10.1109/JSSC.2022.3214064.

[21] D. Shin, J. Lee, J. Lee, J. Lee, H.-J. Yoo, Dnpu: An energy-efficient deep-learning processor with heterogeneous multi-core architecture, IEEE Micro 38 (5) (2018) 85–93. doi:10.1109/MM.2018.053631145.

[22] A. Gonzalez, J. Zhao, B. Korpan, H. Genc, C. Schmidt, J. Wright, A. Biswas, A. Amid, F. Sheikh, A. Sorokin, S. Kale, M. Yalamanchi, R. Yarlagadda, M. Flannigan, L. Abramowitz, E. Alon, Y. S. Shao, K. Asanović, B. Nikolić, A 16mm2 106.1 gops/w heterogeneous risc-v multi-core multi-accelerator soc in low-power 22nm finfet, in: ESSCIRC 2021 - IEEE 47th European Solid State Circuits Conference (ESSCIRC), 2021, pp. 259–262. doi:10.1109/ESSCIRC53450.2021.9567768.

[23] C. Frenkel, M. Lefebvre, D. Bol, Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks, Frontiers in Neuroscience 15 (2021). doi:10.3389/fnins.2021.629892.
URL https://www.frontiersin.org/articles/10.3389/fnins.2021.629892

[24] H. Ren, D. Anicic, T. A. Runkler, TinyOl: Tinyml with online-learning on microcontroller, Arxiv Preprint arXiv:2103.08295 (2021).

[25] V. Ramanathan, Online on-device mcu transfer learning (2020).
URL https://vikramramanathan.com/files/CS249R_Final_Report_Transfer_Learning.pdf

[26] D. Nadalini, M. Rusci, G. Tagliavini, L. Ravaglia, L. Benini, F. Conti, Pulp-trainlib: Enabling on-device training for risc-v multi-core mcus through performance-driven autotuning, in: A. Orailoglu, M. Reichenbach, M. Jung (Eds.), Embedded Computer Systems: Architectures, Modeling, and Simulation, Springer International Publishing, Cham, 2022, pp. 200–216.

[27] C. Cioflan, L. Cavigelli, M. Rusci, M. De Prado, L. Benini, Towards on-device domain adaptation for noise-robust keyword spotting, in: 2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS), 2022, pp. 82–85. doi:10.1109/AICAS54282.2022.9869990.

[28] L. Ravaglia, M. Rusci, D. Nadalini, A. Capotondi, F. Conti, L. Benini, A tinyml platform for on-device continual learning with quantized latent replays, IEEE Journal on Emerging and Selected Topics in Circuits and Systems 11 (4) (2021) 789–802. doi:10.1109/JETCAS.2021.3121554.

[29] D. Rossi, et al., Vega: A Ten-Core SoC for IoT Endnodes With DNN Acceleration and Cognitive Wake-Up From MRAM-Based State-Retentive Sleep Mode, IEEE Journal of Solid-State Circuits 57 (1) (2021) 127–139.

[30] Y. Zhao, C. Liu, Z. Du, Q. Guo, X. Hu, Y. Zhuang, Z. Zhang, X. Song, W. Li, X. Zhang, L. Li, Z. Xu, T. Chen, Cambricon-q: A hybrid architecture for efficient training, in: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), 2021, pp. 706–719. doi:10.1109/ISCA52012.2021.00061.

[31] B. Noune, P. Jones, D. Justus, D. Masters, C. Luschi, 8-bit Numerical Formats for Deep Neural Networks, Arxiv Preprint arXiv:2206.02915 (2022).

[32] J. Park, S. Lee, D. Jeon, A neural network training processor with 8-bit shared exponent bias floating point and multiple-way fused multiply-add trees, IEEE Journal of Solid-State Circuits 57 (3) (2022) 965–977. doi:10.1109/JSSC.2021.3103603.

[33] S. Fox, S. Rasoulinezhad, J. Faraone, david boland, P. Leong, A block minifloat representation for training deep neural networks, in: International Conference on Learning Representations, 2021.
URL https://openreview.net/forum?id=6zaTwpNSsQ2

[34] N. Wang, J. Choi, D. Brand, C.-Y. Chen, K. Gopalakrishnan, Training deep neural networks with 8-bit floating point numbers, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18, Curran Associates Inc., Red Hook, NY, USA, 2018, p. 7686–7695.

[35] A. Agrawal, S. K. Lee, J. Silberman, M. Ziegler, M. Kang, S. Venkataramani, N. Cao, B. Fleischer, M. Guillorn, M. Cohen, S. Mueller, J. Oh, M. Lutz, J. Jung, S. Koswatta, C. Zhou, V. Zalani, J. Bonanno, R. Casatuta, C.-Y. Chen, J. Choi, H. Haynie, A. Herbert, R. Jain, M. Kar, K.-H. Kim, Y. Li, Z. Ren, S. Rider, M. Schaal, K. Schelm, M. Scheuermann, X. Sun, H. Tran, N. Wang, W. Wang, X. Zhang, V. Shah, B. Curran, V. Srinivasan, P.-F. Lu, S. Shukla, L. Chang, K. Gopalakrishnan, 9.1 a 7nm 4-core ai chip with 25.6tflops hybrid fp8 training, 102.4tops int4 inference and workload-aware throttling, in: 2021 IEEE International Solid- State Circuits Conference (ISSCC), Vol. 64, 2021, pp. 144–146. doi:10.1109/ISSCC42613.2021.9365791.

[36] S. Venkataramani, V. Srinivasan, W. Wang, S. Sen, J. Zhang, A. Agrawal, M. Kar, S. Jain, A. Mannari, H. Tran, Y. Li, E. Ogawa, K. Ishizaki, H. Inoue, M. Schaal, M. Serrano, J. Choi, X. Sun, N. Wang, C.-Y. Chen, A. Allain, J. Bonano, N. Cao, R. Casatuta, M. Cohen, B. Fleischer, M. Guillorn, H. Haynie, J. Jung, M. Kang, K.-h. Kim, S. Koswatta, S. Lee, M. Lutz, S. Mueller, J. Oh, A. Ranjan, Z. Ren, S. Rider, K. Schelm, M. Scheuermann, J. Silberman, J. Yang, V. Zalani, X. Zhang, C. Zhou, M. Ziegler, V. Shah, M. Ohara, P.-F. Lu, B. Curran, S. Shukla, L. Chang, K. Gopalakrishnan, Rapid: Ai accelerator for ultra-low precision training and inference, in: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), 2021, pp. 153–166. doi:10.1109/ISCA52012.2021.00021.

[37] J. Oh, S. K. Lee, M. Kang, M. Ziegler, J. Silberman, A. Agrawal, S. Venkataramani, B. Fleischer, M. Guillorn, J. Choi, W. Wang, S. Mueller, S. Ben-Yehuda, J. Bonanno, N. Cao, R. Casatuta, C.-Y. Chen, M. Cohen, O. Erez, T. Fox, G. Gristede, H. Haynie, V. Ivanov, S. Koswatta, S.-H. Lo, M. Lutz, G. Maier, A. Mesh, Y. Nustov, S. Rider, M. Schaal, M. Scheuermann, X. Sun, N. Wang, F. Yee, C. Zhou, V. Shah, B. Curran, V. Srinivasan, P.-F. Lu, S. Shukla, K. Gopalakrishnan, L. Chang, A 3.0 tflops 0.62v scalable processor core for high compute utilization ai training and inference, in: 2020 IEEE Symposium on VLSI Circuits, 2020, pp. 1–2. doi:10.1109/VLSICircuits18222.2020.9162917.

[38] J. Lee, J. Lee, D. Han, J. Lee, G. Park, H.-J. Yoo, 7.7 lnpu: A 25.3tflops/w sparse deep-neural-network learning processor with fine-grained mixed precision of fp8-fp16, in: 2019 IEEE International Solid- State Circuits Conference - (ISSCC), 2019, pp. 142–144. doi:10.1109/ISSCC.2019.8662302.

[39] J. Heo, J. Kim, S. Lim, W. Han, J.-Y. Kim, T-pim: An energy-efficient processing-in-memory accelerator for end-to-end on-device training, IEEE Journal of Solid-State Circuits 58 (3) (2023) 600–613. doi:10.1109/JSSC.2022.3220195.

[40] H. Bian, J. Huang, L. Liu, D. Huang, X. Wang, Albus: A method for efficiently processing spmv using simd and load balancing, Future Generation Computer Systems 116 (2021) 371–392. doi:https://doi.org/10.1016/j.future.2020.10.036.
URL https://www.sciencedirect.com/science/article/pii/S0167739X2033020X

[41] S. Kim, J. Lee, S. Kang, D. Han, W. Jo, H.-J. Yoo, Tsunami: Triple sparsity-aware ultra energy-efficient neural network training accelerator with multi-modal iterative pruning, IEEE Transactions on Circuits and Systems I: Regular Papers 69 (4) (2022) 1494–1506. doi:10.1109/TCSI.2021.3138092.

[42] Y. Wang, Y. Qin, D. Deng, J. Wei, T. Chen, X. Lin, L. Liu, S. Wei, S. Yin, Trainer: An energy-efficient edge-device training processor supporting dynamic weight pruning, IEEE Journal of Solid-State Circuits 57 (10) (2022) 3164–3178. doi:10.1109/JSSC.2022.3174411.

[43] M. Anders, H. Kaul, S. Mathew, V. Suresh, S. Satpathy, A. Agarwal, S. Hsu, R. Krishnamurthy, 2.9tops/w reconfigurable dense/sparse matrix-multiply accelerator with unified int8/inti6/fp16 datapath in 14nm tri-gate cmos, in: 2018 IEEE Symposium on VLSI Circuits, 2018, pp. 39–40. doi:10.1109/VLSIC.2018.8502333.

[44] G. Mehrooz, P. Schneider-Kamp, Optimal path planning for drone inspections of linear infrastructures, in: International Conference on Geographical Information Systems Theory, Applications and Management, 2020.

[45] S. Suk, Minsoo ; Sull, Curvilinear feature extraction and approximations, Computer Vision, Graphics, and Image Processing. (1983) 400–411doi:10.1016/0734-189X(84)90221-4.

[46] Y. Tortorella, L. Bertaccini, D. Rossi, L. Benini, F. Conti, Redmule: A compact fp16 matrix-multiplication accelerator for adaptive deep learning on risc-v-based ultra-low-power socs, in: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2022, pp. 1099–1102. doi:10.23919/DATE54114.2022.9774759.

[47] A. Pedram, R. A. van de Geijn, A. Gerstlauer, Codesign tradeoffs for high-performance, low-power linear algebra architectures, IEEE Transactions on Computers 61 (12) (2012) 1724–1736. doi:10.1109/TC.2012.132.

[48] F. Conti, P. D. Schiavone, L. Benini, Xnor neural engine: A hardware accelerator ip for 21.6-fj/op binary neural network inference, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37 (11) (2018) 2940–2951. doi:10.1109/TCAD.2018.2857019.

[49] A. Garofalo, Y. Tortorella, M. Perotti, L. Valente, A. Nadalini, L. Benini, D. Rossi, F. Conti, Darkside: A heterogeneous risc-v compute cluster for extreme-edge on-chip dnn inference and training, IEEE Open Journal of the Solid-State Circuits Society 2 (2022) 231–243. doi:10.1109/OJSSCS.2022.3210082.

[50] S. Mach, F. Schuiki, F. Zaruba, L. Benini, Fpnew: An open-source multi-format floating-point unit architecture for energy-proportional transprecision computing, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 29, no. 4, pp. 774-78 (2020).

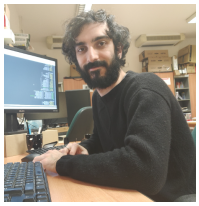
[51] C. R. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Király, P. Montino, D. Kanter, S. A. Ahmed, D. Pau, U. Thakker, A. Torrini, P. Warden, J. Cordaro, G. D. Guglielmo, J. M. Duarte, S. Gibellini, V. Parekh, H. Tran, N. Tran, N. Wenxu, X. Xuesong, Mlperf tiny benchmark, ArXiv abs/2106.07597 (2021).

[52] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition (2016) 770–778doi:10.1109/CVPR.2016.90.

[53] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520. doi:10.1109/CVPR.2018.00474.

[54] A. Burrello, M. Scherer, M. Zanghieri, F. Conti, L. Benini, A microcontroller is all you need: Enabling transformer execution on low-power iot endnodes, in: 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS), 2021, pp. 1–6. doi:10.1109/COINS51742.2021.9524173.

[55] M. v. Baalen, A. Kuzmin, S. S. Nair, Y. Ren, E. Mahurin, C. Patel, S. Subramanian, S. Lee, M. Nagel, J. Soriaga, T. Blankevoort, FP8 versus INT8 for efficient deep learning inference, Arxiv Preprint arXiv:2303.17951v1 (2023).



Yvan Tortorella received his Master's Degree in Electronic Engineering in October 2021 from the University of Bologna. He is currently pursuing a Ph. D. in Digital Systems Design in the group of Professor Luca Benini at the Department of Electrical and Information Engineering (DEI) of the University of Bologna. His research interests include the design of PULP (Parallel Ultra-Low Power)-based hardware accelerators for ultra-low power Machine Learning and the design of RISC-V-based computer architectures for satellite applications.

Luca Bertaccini received the M.Sc. degree in Electronic Engineering from the University of Bologna in 2020. He is currently pursuing a Ph.D. degree at ETH Zürich in the Digital Circuits and Systems group led by Prof. Luca Benini. His research interests include heterogeneous systems-on-chip, energy-efficient hardware accelerators, computer arithmetic, and transprecision computing. He received the 2021 IEEE ASAP Best Paper Honorable Mention.

Luca Benini holds the chair of digital Circuits and systems at ETHZ and is Full Professor at the Università di Bologna. He received a PhD from Stanford University. Dr. Benini's research interests are in energy-efficient parallel computing systems, smart sensing micro-systems and machine learning hardware. He has published more than 1000 peer-reviewed papers and five books. He is a Fellow of the IEEE, of the ACM and a member of the Academia Europaea. He received the IEEE Mac Van Valkenburg award in 2016 and the ACM/IEEE A. Richard Newton Award in 2020.

Davide Rossi received the Ph.D. degree from the University of Bologna, Bologna, Italy, in 2012. He has been a Post-Doctoral Researcher with the Department of Electrical, Electronic and Information Engineering "Guglielmo Marconi," University of Bologna, since 2015, where he is currently an Associate Professor. His research interests focus on energy-efficient digital architectures. In this field, he has published more than 100 papers in international peer-reviewed conferences and journals. He is recipient of Donald O. Pederson Best Paper Award 2018, 2020 IEEE TCAS Darlington Best Paper Award, 2020 IEEE TVLSI Prize Paper Award.

Francesco Conti received the Ph.D. degree in electronic engineering from the University of Bologna, Italy, in 2016. He is currently a Tenure-Track Assistant Professor with the DEI Department, University of Bologna. From 2016 to 2020, he held a research grant with the University of Bologna and a Post-Doctoral Researcher with ETH Zürich. His research is centered on hardware acceleration in ultra-low power and highly energy efficient platforms, with a particular focus on System-on-Chips for Artificial Intelligence applications. His research work has resulted in more than 70 publications in international conferences and journals and was awarded several times, including the 2020 IEEE TCAS-I Darlington Best Paper Award.