RedMule：面向灵活、高能效片上线性代数及 TinyML 训练加速的混合精度矩阵-矩阵运算引擎

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

[a]博洛尼亚大学，意大利博洛尼亚[b]苏黎世联邦理工学院，瑞士苏黎世

arXiv:2301.03904v2 [cs.AR] 6 May 2023

摘要

人们对 TinyML 的兴趣日益浓厚。TinyML 即在数十毫瓦的功耗预算下进行近传感器机器学习，目前这一趋势正推动 TinyML 级训练的发展，而不再局限于推理。当前的训练算法基于各种形式的误差反向传播和梯度反向传播，为满足精度和动态范围要求，依赖浮点矩阵运算。迄今为止，这些运算的能耗和功耗成本一直被认为对于 TinyML 场景过高。本文针对数毫瓦功耗预算下的近传感器训练这一尚待解决的挑战，介绍 RedMulE（Reduced-Precision Matrix Multiplication Engine，缩减精度矩阵乘法引擎）。这是一款面向多精度浮点通用矩阵-矩阵运算（GEMM 运算）加速而设计的低功耗专用加速器，支持 FP16 以及混合FP8格式，其中包含 {符号位、指数、尾数} = （$\{1,4,3\}$，$\{1,5,2\}$）。我们将 RedMulE 集成到并行超低功耗（PULP）集群中，该集群包含八个共享紧耦合数据存储器的高能效 RISC-V 核心，并采用 22 纳米工艺实现了整个系统。在最佳效率点（@ 470 MHz，0.65 V），配备 RedMulE 的 PULP 集群在常规通用矩阵-矩阵乘法（GEMM）期间可实现 755 GFLOPS/W 和 920 GFLOPS/W；执行 GEMM 运算时，针对 FP16 和 FP8 输入/输出张量，分别可达 1.19 TFLOPS/W 和 1.67 TFLOPS/W。在最佳性能点（@ 613 MHz，0.8 V），RedMulE 的性能最高可达 FP16 的 58.5 GFLOPS 和 FP8 的 117 GFLOPS，计算单元阵列利用率为 99.4%，平均功耗低于 60 mW。因此，它能够在 TinyML 应用场景中实现深度学习模型的设备端训练，同时灵活高效地处理其他常见类型的线性代数问题。

关键词：通用矩阵-矩阵乘法、通用矩阵-矩阵运算、硬件加速器、嵌入式系统、在线学习、TinyML。

1. 引言

过去几年，连接到网络并运行机器学习（ML），尤其是基于深度学习（DL）的算法（如深度神经网络（DNN））的物联网（IoT）设备数量大幅增加。为了减少通过网络传输的数据量、提高能效并避免网络拥塞，计算任务日益从数据中心转移到功耗预算较低（平均几毫瓦、峰值一百毫瓦）的高能效物联网终端节点 [1],，由此推动了Tiny-ML领域的研究与应用。

极端边缘应用（如神经网络（NN）的训练与推理、图分析与处理、 [2,3], 最短距离问题 [4], 以及基于模型的控制）都依赖通用矩阵-矩阵乘法（GEMM）或通用矩阵-矩阵运算（GEMM-Ops）这一最重要的核函数。GEMM-Ops 是与 GEMM 结构相同、但以其他映射和归约运算替代规范乘加的运算。由于这些计算模式具有相似性，近期有人提出 [6] 为 Tensor Cores 增加对 GEMM-Ops 的支持，从而将其加速能力拓展到更广泛的应用领域。目前，针对超低功耗嵌入式系统的同类工作仍然缺乏。

在台式机、移动设备和数据中心计算中，通常采用单精度和双精度浮点（FP）运算来处理深度学习和线性代数应用，以可接受的面积和能耗开销实现较高精度。然而，在嵌入式设备上，功耗和面积约束要严格得多。近年来，人们投入了大量精力，将基于线性代数的算法以及在线学习 [7] 适配到低精度格式，例如 FP16 [8, 9] 和 FP 8 [10, 11],，同时将精度损失控制在较小范围内。这些算法进展带来了性能和能效提升 [12, 13],，为在智能可穿戴设备等极边缘计算系统上部署深度学习模型的持续学习和自适应开辟了道路。然而，这类设备通常使用的微控制器单元（MCU）计算能力有限，尤其是在执行 FP 运算方面。

本文介绍了 RedMulE（低精度矩阵乘法引擎），这是首款面向 TinyML 的开源

电子邮箱地址： yvan.tortorella@unibo.it (Yvan Tortorella), lbertaccini@iis.ee.ethz.ch(Luca Bertaccini),lbenini@iis.ee.ethz.ch (LucaBenini), davide.rossi@unibo.it(Davide Rossi), f.conti@unibo.it (Francesco Conti)

硬件加速器，支持混合浮点精度（Hybrid-FP8 和 FP16）线性代数，并紧密集成于基于 RISC-V 的并行超低功耗（PULP）集群中。RedMulE 高度参数化，支持可在设计时调节的不同配置。它与集群核心共享 L1存储器，从而实现硬件加速器与核心之间的细粒度协作。RedMulE 可加速低精度GEMM（神经网络训练算法背后的关键核函数），使超低功耗片上系统（SoC）能够实现片上学习。此外，它还支持更广泛的一类操作，称为通用矩阵乘法运算 [6]，因此适用于广泛的应用。

我们在一个 22 nm CMOS技术的 8 核 PULP 集群中对设计进行了原型验证，并实例化了一个包含 48 个内部计算单元的 RedMulE 实例。RedMulE 仅占用 0.15 mm^2，，相当于整个集群面积的 24%。与在 RISC-V 核心上并行执行相比，它在常规 FP16 GEMM 中的加速比最高可达 15×，在 GEMM 操作中的加速比最高可达 62×；在 613 MHz、0.8 V 下，性能最高可达 58.5 GFLOPS（计算单元利用率为 99.4%）。在最佳效率工作点，即 470 MHz、0.65 V 下，RedMulE 在 GEMM 和 GEMM 操作中的能效分别最高可达 772 GFLOPS/W 和 1.19 TFLOPS/W，同时性能达到 44.8 GFLOPS。采用 FP8 输入/输出张量表示进行计算时，包含 96 个计算单元的 RedMulE 实现版本在 613 MHz、0.8 V 下的性能最高可达 117 GFLOPS，能效最高可达 920 GFLOPS/W，与 INT8推理加速器相当。

2. 相关工作

人们对执行基于线性代数的算法（如神经网络推理和训练）表现出浓厚兴趣，这推动了针对这类任务的各类硬件平台的发展，其范围从数据中心计算系统到超低功耗嵌入式平台 [14]。英伟达最新的 Hopper H100 [15]图形处理器（GPU）是面向深度学习任务（如神经网络推理和训练）的数据中心计算平台中最具代表性的例子。H100 的性能达到 1978 TFLOPS，功耗为 700W，并可使用低位宽 FP8格式训练 Transformer模型等大型神经网络模型。

另一方面，由于受到严格的功耗、能耗和成本限制，要在智能可穿戴系统等极端边缘设备的超低功耗 TinyML片上系统上运行基于深度学习的算法颇具挑战。在实际应用中，极端边缘推理是可行的，因为它可以采用低精度整数运算，从而减少模型内存占用，并在精度损失有限的情况下提升底层架构的能效 [16, 17]。相反，极端边缘神经网络训练需要大量存储器，并且需要进行浮点计算，通常会导致功耗超出TinyML的限制 [17, 18]。本节聚焦于强调中等功耗边缘训练的嵌入式平台。

2.1. 推理加速器

专为低功耗深度学习推理设计的硬件加速器，为基于软件的执行方案提供了颇具吸引力的替代选择 [17, 19]。Diana [20], 是一款低功耗神经网络片上系统，其特点是配备了数字神经网络推理加速器和模拟存内计算核心，两者集成于共享存储器子系统中，且仅支持窄整数格式。DNPU [21] 是一款全数字高能效深度学习处理器，采用65纳米工艺和支持16位定点运算的异构架构，专为卷积神经网络和递归神经网络推理加速而设计。Gemmini [22]是一款 16×16脉动式加速器，专为深度神经网络推理设计，配备8位乘加单元，并支持运行时可编程的权重驻留和输出驻留数据流。

2.2. 设备端学习

设备端学习是一项新兴且尚待解决的挑战，涉及在超低功耗通用微控制器上训练深度学习模型。为此，许多研究探索了直接反馈对齐或平衡传播等算法。然而，由于存在严重的收敛困难，这些方法已被证明不如经典反向传播方法有效 [23]。TinyOL [24] 和 [25] 专注于使用基于 Cortex-M内核的低成本 Arduino Nano 微控制器训练神经网络。另一方面，PULP Trainlib [26], Cioflan 等人 [27], 和 Ravaglia 等人 [28]提出的方法均旨在实现设备端学习和自适应，可运行于 Vega [29], 等 RISC-V多核 PULP集群，并支持混合浮点精度，范围从 IEEE 754标准 FP32和 FP16 到*bfloat*。然而，超低功耗微控制器通常速度较慢且可用浮点单元数量有限，这限制了这些库的性能。

2.3. 训练加速器

为解决低功耗处理器上软件库所能实现的训练性能有限的问题，一些研究人员转而采用硬件加速 [14]。

寒武纪-Q [30]是一款面向训练的芯片，基于8位定点运算，实现了高精度和高能效。然而，许多常见训练算法需要浮点运算才能确保收敛 [31, 32, 33, 34]。大多数采用浮点运算的面向训练芯片，其功耗范围都不适用于极端边缘应用。IBM 提出了一种人工智能计算平台 [35, 36], ，其混合精度引擎 8×8 支持 FP16 和混合FP8训练，而 [37] 仅支持 FP16 和 FP32。类似地，LNPU [38]支持混合 8 位和 16 位浮点精度，可用于片上训练。尽管这些芯片在神经网络训练期间的功耗远低于数据中心GPU（即仅数瓦，而非数百瓦），但仍无法满足 TinyML设备数十毫瓦的功耗约束。

近年来，已有一些面向训练的 SoC 问世，能够满足极端边缘应用的功耗预算。T-PIM [39] 是一款采用 28 nm 工艺、用于设备端学习的存内处理加速器。训练时，在稀疏度为 0% 的情况下，其能效最高可达 250 十亿次运算每秒每瓦

表1：RedMulE支持的通用矩阵-矩阵运算集合

$Z = (X \circ W) \star Y$				
组	**核函数**	**○**	**★**	**Res**
	矩阵乘法	×	+	$Z = (X \times W) + Y$
Group 1	最大值 关键路径	+	*max*	$Z = \text{最大值}[Y,(X + W)]$
	全对 最短路径	+	*min*	$Z = \text{最小值}[Y,(X + W)]$
	最大值 可靠性路径	×	*max*	$Z = \text{最大值}[Y,(X \times W)]$
	最小值 可靠性路径	×	*min*	$Z = min[Y,(X \times W)]$
Group 2	最小 生成树	*max*	*min*	$Z = min[Y,\ max\ (X,\ W)\ ]$
	最大值 容量路径	*min*	*max*	$Z = \text{最大}[Y,\text{最小}(X, W)]$

，工作频率为 280 MHz，且功耗不超过 51.23 mW。然而，T-PIM 以及近期提出的所有存内处理方案均不支持浮点计算，因此不适用于标准反向传播。为了在较低功耗预算下支持神经网络训练，许多面向训练的芯片都广泛采用剪枝，以在训练过程中提高稀疏度 [40], , 但缺乏通用性。例如，TSUNAMI [41] 和 Trainer [42] 都是面向极端边缘神经网络推理与训练设计的加速器，通过采用剪枝和零值跳过来满足 TinyML 的功耗约束。Anders 等人 [43] 提出了一款用于稠密-稀疏矩阵乘法的可重构加速器，支持混合精度计算，适用于面向训练的应用；该加速器支持 FP16 乘法和 FP32 累加，同时面积占用小、能效高。然而，这种加速器不具备参数化能力，因此在需要更高性能时，无法在设计时快速扩展。此外，其在实际系统中的集成尚未经过评估，而且不支持压缩 FP8 输入/输出张量，而这类张量有助于在存储器资源有限的边缘设备上训练更大的神经网络模型。

2.4. 通用矩阵乘法运算芯片

迄今为止提到的所有面向训练和推理的芯片示例，都只针对最常见的深度学习运算（例如矩阵乘法和卷积）。然而，大量核函数与通用矩阵乘法具有相同的计算结构，但不依赖乘法和加法作为基本运算，因此属于通用矩阵乘法运算的范畴。图分析应用，例如广度优先搜索 [2, 3], 最短距离问题 [4]（常用于嵌入式无人机导航中的路径规划优化 [44], ）以及用于计算机视觉的最小生成树 [45], , 都是运用通用矩阵乘法运算的应用示例。SIMD

[2] [6]通过在 NVIDIA 流式多处理器架构之上构建用于通用矩阵乘法操作加速的功能单元，解决了这一问题；该设计模仿 Tensor Cores 的结构，并提供专用的指令集架构扩展。该设计采用 45 纳米工艺实现。相较于在 CUDA 核心上执行相同核函数，将所有 SIMD[2] 扩展添加到基线矩阵乘法单元后，加速比最高可达 15.8× ，代价是增加 69% 的开销

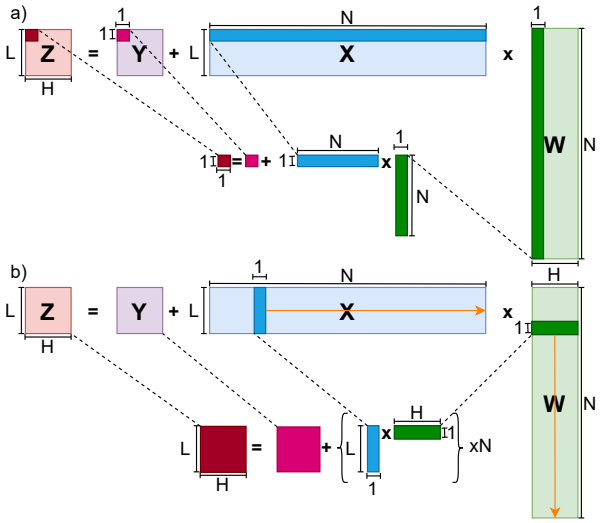


图 1：通过 a）标量点积微内核和 b）块点积（或外积）微内核执行通用矩阵乘法。

的面积开销。

本文提出了扩展版 Red- MulE [46]，它具备以下独特的功能组合：

- 一种基于浮点单元的计算元件（计算单元）阵列，用于在嵌入式片上系统上对通用深度学习模型进行高效训练和推理，并额外支持低位宽浮点计算。我们将 RedMulE 与 RISC-V 处理器的并行集群紧密耦合，以实现采用复杂训练算法时的最大灵活性；

- 支持通用矩阵乘法运算，同时相较于仅包含 GEMM 的实现仅增加较低的面积开销（16%），从而适用于更广泛的应用；

- 一种完全参数化的设计，支持实例化 f 各种计算元件阵列、内部缓冲区和存储器接口配置。

3. 背景

3.1. 广义矩阵-矩阵运算

在本工作中，我们将广义矩阵-矩阵运算（*GEMM-Ops*）定义为所有具有$f2(\mathbf{Y}, f1(\mathbf{X}, \mathbf{W}))$形式的运算，具体可表示为：

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

其中，$\circ$ 对应$f1()$，$\star$ 对应$f2()$。表1展示了一些通用矩阵乘法运算的示例，并将其分为两组。第1组包含所有 $\circ$ 运算符可属于 +/×类型，而 $\star$ 可以是*min*/*max*的通用矩阵乘法运算。第2组包含 $\circ$ 运算符也属于*min*/*max* 类型的通用矩阵乘法运算内核。$\mathbf{X}$ 是一个大小为$M \times N$的矩阵，$\mathbf{W}$是一个大小为$N \times K$的矩阵，而$\mathbf{Z}$ 和$\mathbf{Y}$ 的大小均为$M \times K$。

GEMM 与通用矩阵乘法运算之间的相似性，使矩阵计算单元很适合扩展以支持通用矩阵乘法运算，从而增强其灵活性，加速广义并行代数运算符。这类算法

也非常适用于机器学习应用，因为矩阵是所有深度学习模型的基线结构。为此，必须注意，公式1的结构是对称的。因此，对于机器学习应用，无需将**X** 或**W** 指定为输入矩阵或权重矩阵，因为它们的角色可以灵活互换。

3.2. 线性代数加速策略的渐近最优性

内存加载/存储操作扩大了理论性能与实际性能和效率之间的差距。因此，最大化每次内存访问所执行的运算数量，即算术强度，是实现高效设计的关键。正如 Pedram [47], 所分析的，标量点积和向量单元并不能保证在每次内存加载/存储访问所执行的运算数量方面实现最佳权衡。如图1a所示，在一个N维数组上运行的简单标量点积执行 $2 \times N$ 次运算（N 次乘法和 $+N$ 次加法）。该核函数执行的内存操作包括：N 次加载X、N 次加载W、一次加载Y 以及一次存储Z。由此得到的算术强度为：

$$Intensity_1D = \frac{OPs}{LD/ST} = \frac{2N}{2N+2} \sim 1, \quad (N \to \infty). \qquad (2)$$

二维$L \times H$ 数组利用块点积（外积）微内核执行 GEMM。考虑一个$L \times H$ 二维数组，它可以处理$L \times 1$ 个和 $1 \times H$ 个向量，每个向量包含N 个元素，如图1b所示。对这两个向量执行的运算为 $2 \times L \times H$，重复N 次。由此产生的加载/存储操作包括：$L \times N$ 次加载X、$H \times N$ 次加载W、$L \times H$ 次加载Y 以及$L \times H$ 次存储Z。经过这些变化后，公式2变为：

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

公式3表明，当$L = H$时，运算次数随二维数组的规模呈二次增长，而内存访问次数仍呈线性增长。这说明，与标量单元或向量单元相比，二维数组的效率更高。因此，我们将在 RedMulE 设计中采用外积方法。

4. 架构

本节介绍 PULP集群、我们所依托的硬件模板，以及 RedMulE 微架构。

*4.1. PULP*集群与 *RedMulE*

图2展示了 PULP集群的架构。这是一种多核架构，包含数量可配置（2–16 个）的 32 位 RISC-V 通用处理器核，并配备部分共享、部分私有的指令缓存。在这项具体工作中，我们重点研究一个包含 8 个 RISC-V 核心，配备 128kB 紧耦合数据存储器（TCDM），并将其划分为 16 个存储体，以实现字级交错，

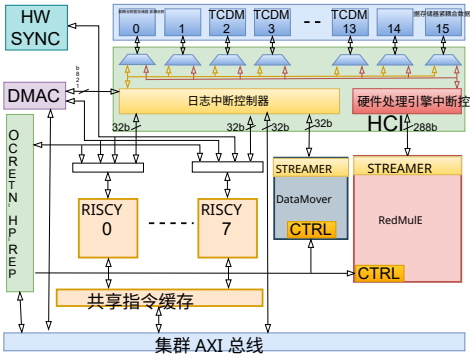


图 2：集成 HWPE 的 PULP 集群架构 n.

从而降低争用程度。PULP集群还配备了一个事件单元，用于灵活的内部同步，以及一个专用的直接内存访问控制器（DMAC），可在 TCDM 与外部存储器之间高效传输数据。外设互连允许 RISC-V 核心对板载外设（如 DMAC）进行编程，而 AXI4 全交叉开关互连则支持与外部环境通信。PULP集群还配备了一个系统级时钟门控单元，在集群未使用时消除其动态功耗。

通过集成专用硬件加速器（称为硬件处理引擎，HWPE），可以进一步增强 PULP集群的能力。HWPE 由 RISC-V 核心通过外设互连进行软件编程，并与 RISC-V 核心和 DMAC 共享紧耦合数据存储器。在这一意义上，HWPE 与集群核心紧密耦合 [48]。核心、DMAC 和加速器通过单周期延迟的异构集群互连（HCI）访问共享的紧耦合数据存储器 [49]。此类互连具有一个对数分支，可实现从核心或 DMAC 等 32位主端口到每个字交织存储体的全互连单周期访问。冲突通过轮询方案管理，每个存储体仅允许一个发起方获得授权。另一个分支是浅分支。该分支具有一个单一的 n 位参数化端口，该端口连接到相邻的 32 位存储器组，这些存储器组被视为一个更宽的存储体，且无需仲裁。该分支便于集成 HWPE 等紧耦合加速器。浅分支端口的位宽可通过参数设置，以满足 HWPE 的需求。TCDM 存储体通过一组多路复用器连接到两个 HCI 分支；多路复用器根据可配置的防饥饿轮转方案授予其中一个分支访问权限，并在 N 个周期内为 HWPE 分配最多连续 K< N 个周期。

在神经网络工作负载的执行过程中，尤其是在推理和训练期间，众所周知，在线数据编组操作会显著降低性能。因此，我们的 PULP 集群配备了 DataMover [49]。DataMover 是一款小型加速器，能够转置存储在紧耦合数据存储器中的三维张量，所需时间比八个 RISC-V 核心少 33%，能效最高可提升 50×（待转置数据块的精度越低，优势就越显著）。该加速器支持可配置的数据元素

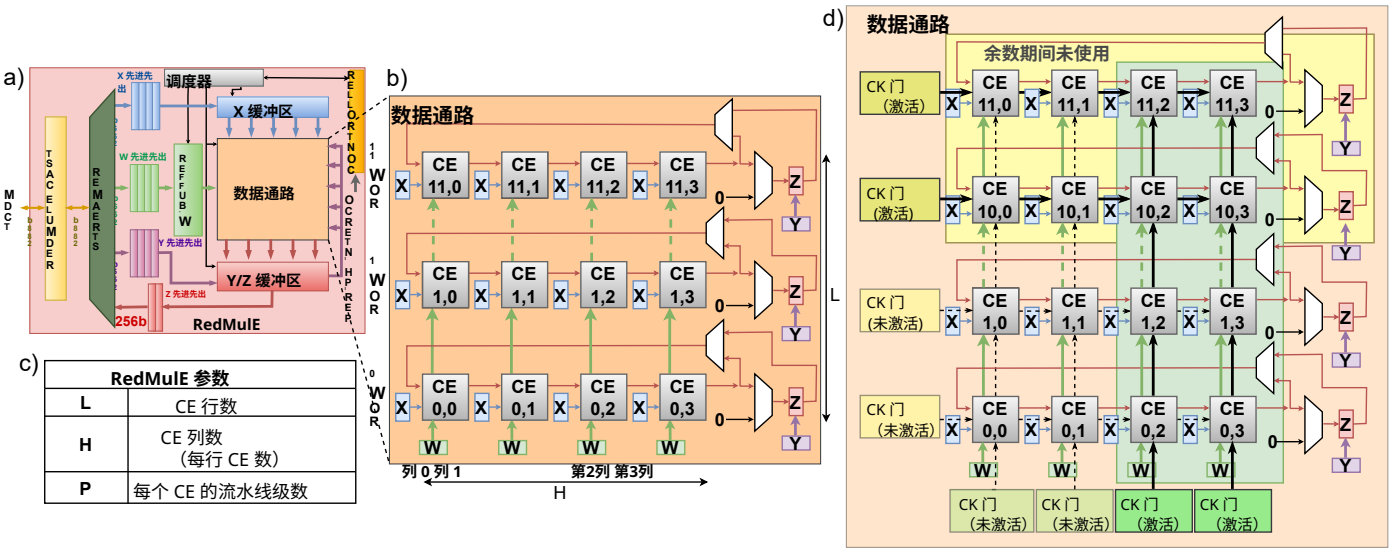


图 3：a）RedMulE 内部架构；b）RedMulE 数据通路微架构；c）RedMulE 设计时可用参数表；d）数据通路级时钟门控逻辑。

位宽，范围从 32 位到 1 位。

4.2. RedMulE

4.2.1. 全局架构

RedMulE 是一种专用处理器，旨在加速通用矩阵乘法运算。其架构如图 3a 所示。RedMulE 的核心是数据通路，由计算单元组成的二维阵列构成，并按图 3b 所示互连。计算单元排列为L 行，每行包含H 列。在每一行中，若干H 个计算单元依次级联，使得每个计算中间乘积的计算单元都将其结果传递给下一个计算单元。每行最后一个计算单元计算出的部分乘积会反馈，作为同一行第一个计算单元的累加输入。RedMulE数据通路的设计时可配置参数包括内部计算单元的数量、每个计算单元的流水线寄存器（P），以及内部计算精度（浮点位宽）。RedMulE 的所有参数均可在设计时调整，汇总见图 3c。

图3d展示了RedMulE内部实现的部分分层时钟门控逻辑。所提出的节能方案采用细粒度时钟门控机制，降低 RedMulE 数据通路中未使用部分（行和列）的动态功耗。计算余数时，RedMulE可以动态禁用不会用到的CE行。CE列也同样如此：RedMulE会根据计算阶段，对数据通路 中相应列的时钟进行门控。在数据通路中，每个CE分为两个阶段，分别用于选择表1中所示的 $\circ$ 和 $\star$ 运算。每个CE都采用经改造的开源*FPnew* 跨精度浮点单元（FPU）[50]中的计算浮点模块，使其内部流水线寄存器能够支持RedMulE运行期间由存储器停顿引起的反压。图4和第4.2.2节将详细介绍所提出的时钟门控方案如何在CE微架构中分层传播，其中专用时钟门控单元负责禁用未使用的单元。

为向数据通路 提供数据，RedMulE按照硬件处理引擎的设计策略 [1]，集成了*Streamer*。*Streamer*是一种专用存储器访问单元，通过一个参数化宽度（32位的倍数）的单一宽端口将RedMulE连接到HCI浅分支，用于加载和存储操作。来自HCI的输入流会传递到一个单输入多输出分发器，该分发器仅将有效信号转发到选定的输出通道；同时，每个输出通道都会将来自HCI的输入流传递到加速器输入端口。另一方面，RedMulE生成的数据流会在写操作期间传递到HCI接口。

该 *Streamer* 连接到三个内部缓冲区：X 缓冲区每隔$H \times (P + 1)$ 个周期改变一列的所有L 个输入；W 缓冲区由H 个移位寄存器组成，每个移位寄存器每个周期向一列中的所有L 个**W**元素广播一个新的**W**元素；Z 缓冲区则用于缓存计算得到的**Z**元素。同一个缓冲区还用于预加载**Y**元素，并将其推入数据通路。由于无需使用单独的缓冲区存储**Y**偏置，该方案节省了加速器的面积和功耗。三个缓冲区均受益于图 3d 所示的时钟门控逻辑。具体而言，在计算输入矩阵和输出矩阵中的余数时，数据通路 的某些行不执行任何计算，因此X 缓冲区和Z 缓冲区中的相应线路也会进行时钟门控。对于 CE 列数以及W 缓冲区中的线路，同样适用这一考虑。如果某一 CE 列由于存在余数而不执行任何计算，则W缓冲区中的相应线路会动态进行时钟门控。

加速器的控制端分为两个子模块，即调度器 和控制器，它们包含供核心访问的寄存器文件，用于配置加速器，并协同调节加速器的执行。

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

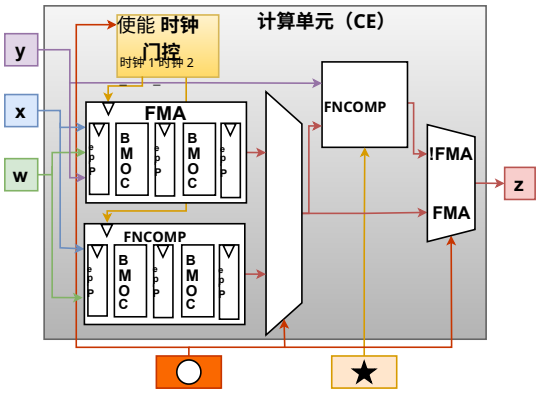


图4：支持 GEMM 操作的 RedMulE 计算单元微架构

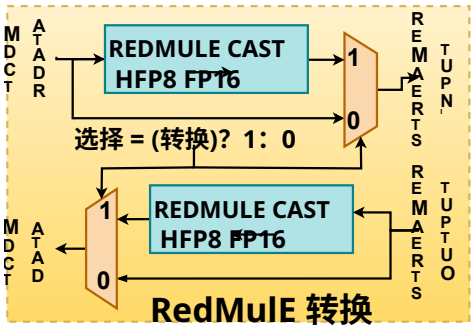


图 5：RedMulE 混合精度支持转换模块的内部架构。

4.2.2. 计算单元的 *GEMM* 操作扩展

我们为 RedMulE 的每个计算单元添加了专用硬件，以支持执行表1中的所有通用矩阵乘法运算。图4展示了每个计算单元的微架构，该架构包含两个阶段。第一阶段选择 $\circ$ 运算，并包含一个融合乘加（FMA）单元和一个浮点非计算操作（FNCOMP）单元，后者实现浮点最小值/最大值操作。**X**和**W** 元素随后被传送至 FMA 和 FNCOMP 模块。根据所需的 $\circ$ 运算，多路复用器会选择 FMA 或 FNCOMP 的结果。同时，图4所示的时钟门控模块会冻结未使用模块的输入操作数，从而彻底消除不必要的开关活动及相关动态功耗。FMA 和 FNCOMP 各自包含P 个流水线寄存器，确保两个模块具有相同的延迟。我们改造了 FMA 和 FNCOMP 模块的流水线，其设计均基于开源*FPNew* 跨精度 FPU [50]，使其能够在 RedMulE 运行期间应对存储器停顿造成的反压。

在每个计算单元中，**Y** 元素会在第一级传递到 FMA 单元，同时也直接连接到计算单元第二级的输入端。每个计算单元的第二级支持 $\star$ 运算选择，并包含一个全组合逻辑 FNCOMP 模块。输出多路复用器可选择第二级 FNCOMP 的输出（用于 GEMM 操作），也可选择第一级 FMA 的输出（用于执行简单的 GEMM 操作）。所提出的架构解决方案能够以紧凑的实现方式执行表1中列出的所有操作，仅添加严格必需的部分。

4.2.3. 混合精度扩展

混合 FP8 精度格式可作为一种高效的压缩方案，使极端边缘设备能够进行深度学习推理和训练。混合 FP8 精度意味着，用于表示张量的{符号位、指数、尾数} 结构可以是 $\{1,5,2\}$ 或 $\{1,4,3\}$[2]。前一种格式最适合梯度的反向传播，因为它具有更大的动态范围，但精度较低。与此同时，后一种格式更适合激活值的前向传播，因为它具有更大的尾数 [11, 10]。虽然 8 位表示能有效减少数据占用空间，但精度累加能力下降可能会严重影响准确度。为支持这一用例，RedMulE 内部采用固定的 FP16 精度，但仍接受压缩的 FP8 格式作为输入，并可生成 FP8 压缩输出张量。为此，我们扩展了 RedMulE 的架构，在 Streamer 与 HCI接口之间增设了一个专用的类型转换模块，如图 3a 所示。

图 5 展示了转换模块的架构实现。该模块包含两个浮点转换单元：输入转换单元将 8 位浮点输入流转换为 16 位浮点数据，送入加速器，使数据通路中的计算单元能够以更高精度运算，从而确保中间累加具有较高精度。计算完成后，输出转换单元会在写入存储器之前，将 RedMulE 生成的 16 位浮点结果转换为 16 位或 8 位编码的输出流。如果输入张量以 16 位表示，则可将转换单元从数据通路中移除。

仅针对深度学习应用场景，RedMulE 也可在设计时配置为仅加载和存储 HFP8 操作数。在这种用例中，采用 8 位格式表示输入和输出张量，可在保持存储器带宽不变的情况下，使存储器读写的元素数量翻倍。因此，每行中的计算单元数量也可翻倍，与使用 16 位输入的情况相比，RedMulE 的性能也随之翻倍。

*4.3. RedMulE*计算模型

图 6a 展示了 RedMulE 如何执行 GEMM 操作，并通过计算矩阵对其进行可视化；图 6b 和图 6d 则展示了计算单元一行内的运算详细顺序，并以此举例说明 GEMM 执行过程。在本讨论中，我们重点关注一种 RedMulE 实现，其特征为 $L = 12$、$H = 4$ 和 $P = 3$。RedMulE 运算首先将 L 行 **Y** 矩阵数据预加载到 Z 缓冲区中，每行包含 $H \times (P + 1) = 16$ 个 FP16 元素（存储器宽度为 256 位／内部精度为 16 位），即第 0 行的 $\mathbf{y}_{0,0}$ - $\mathbf{y}_{0,15}$、第 1 行的 $\mathbf{y}_{1,0}$ - $\mathbf{y}_{1,15}$，依此类推。随后，RedMulE 也按照相同模式预加载 X 缓冲区，并

[2]也称为英伟达的 E4M3 和 E5M2 [https://tinyurl.com/mkhbxj3v]

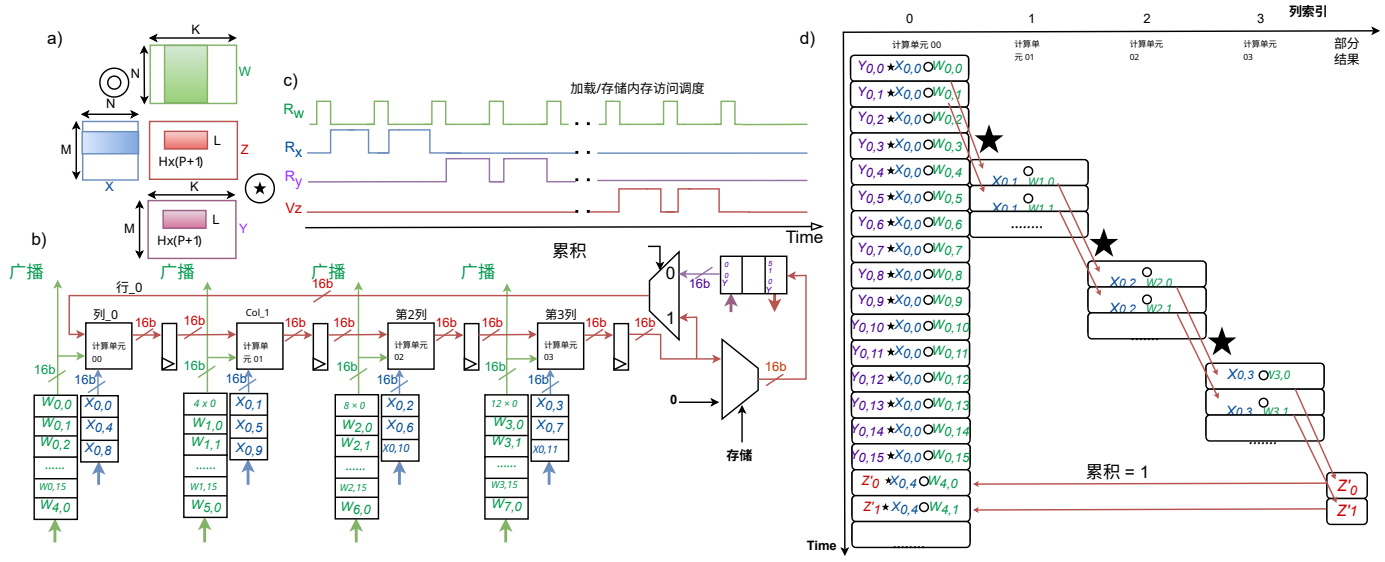


图 6：a) GEMM 操作在矩阵上的执行分 布；b) RedMulE 数据通路中的计算单元行；c) 文中所述加载/存储模式下的内存访问调度关于 R（就绪）和 V（有效）握 手信号；d）计算单元行内的流水线演进。

接着将一组 $H \times (P+1) = 16$ **W** 个元素（$\mathbf{w}_{0,0}$ - $\mathbf{w}_{0,15}$）加载到 W 缓冲区的第一个移位寄存器中。每个 **W** 元素都会广播到第一列数据通路中的所有 L 个计算单元。在 **W** 元素广播的同时，Z 缓冲区逐周期将 **Y** 元素送入计算元件阵列，以便在执行 $\circ$ 运算的过程中执行 $\star$ 运算。

经过 $P+1$ 个周期后，第一列中的每个 L 计算单元都会将其计算出的部分结果转发给第二列中的相邻计算单元。加速器会加载另一组 $H \times (P+1)$**W** 个元素（$\mathbf{w}_{1,0}$ - $\mathbf{w}_{1,15}$），并将其广播给第二列中的所有计算单元。当一行中的所有 H 个计算单元都完成计算，得出一组 $H \times (P+1)$ 个行列中间结果后，RedMulE 会启用反馈（累积 = 1），将中间结果提供给该行首个计算单元的累加输入，然后重复计算。随后，*Streamer* 会立即将下一个 **Y** 子矩阵重新加载到 Z 缓冲区中，使其为下一次计算做好准备。在 Z 缓冲区重新加载期间，X 缓冲区会向第一列计算单元提供新的 **X** 操作数，并将一组新的 $H \times (P+1)$ 个 **W** 元素重新加载到第一个 W 移位寄存器中。经过 $(P+1)$ 个周期后，第一列中的所有 L 个计算单元都会生成新的部分乘积，并将其提供给第二列中的计算单元。X 缓冲区会向第二列计算单元的输入端提供新的 **X** 操作数，而 W 缓冲区则会将一组新的 $H \times (P+1)$ 个 **W** 元素加载到第二个 W 移位寄存器中以供广播，随后计算继续进行。图 6d 展示了从 GEMM 操作开始到部分结果重用（累积 = 1）时，计算单元行内流水线中的数据详细序列。

为保证加速器中的数据流持续不断，W-缓冲区每隔（$P+1$）个周期访问一次存储器，以加载一组新的$H \times (P+1)$个**W**元素。当X缓冲区和Z缓冲区为空时，RedMulE会重新使用*Streamer* 端口来加载**X** 和**Y**操作数。具体做法是在两次相邻的**X** 或**Y** 矩阵访问之间交错进行存储器访问

W矩阵访问，直至X和Z缓冲区完全填满。图 6c 展示了对不同矩阵的存储器访问如何交错进行，并通过就绪（R）和有效（V）握手信号描述存储器访问。*Streamer*加载和存储单元完全支持反压，其机制基于R/V 握手信号。该机制使存储器访问及数据的消耗/生成与*Datapath*完全解耦。加载操作的V 信号和存储操作的R 信号由*Streamer* 自身生成，仅取决于存储器停顿；这些停顿可以通过FIFO元素加以缓解，而不受*Datapath*实际使用情况的影响。另一方面，如图 6c 所示，*Datapath* 使用加载操作的R 信号和存储操作的V 信号来控制存储器访问顺序，并通过交错访问来维持持续的数据流。这样设计是为了最大限度地提高内存端口利用率，因为采用单个内存端口也有助于减小Streamer的整体面积。

完成整个行列运算后，Z 缓冲区会暂存最终的子矩阵。随后，在两次相邻的**W** 加载访问之间交错执行存储操作，直到 Z 缓冲区清空，并可重新加载**Y**元素。通过这种方式，RedMulE 利用单个宽内存端口优化带宽利用率，并实现最高 99.4% 的计算单元利用率。

5. 实现与测量

5.1. 实验设置

我们的实验重点关注一个 RedMulE_{12x4} 实例，其$H = 4$、$L = 12$、$P = 3$，包含 48 个计算单元和一个 288 位宽的 HCI 端口，用于 256-bit + 32 位非字对齐访问。我们还研究了一个 RedMulE_{12x8}，因为如第 5.2.3 节所述，它使用相同的内存接口，但计算单元数量是其两倍。

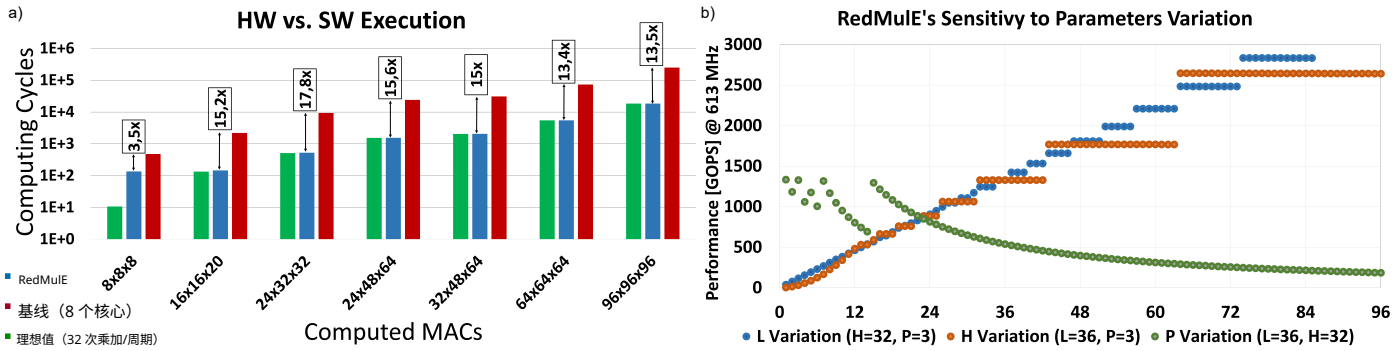


图7：a）RedMulE 综合后的通用矩阵乘法执行与在 8 个 RISC-V 核心上运行的软件相比；b）RedMulE 对 L、H 和 P 参数的敏感性

我们的实验采用格芯 22nm 工艺，并使用 Synopsys Design Compiler 进行综合（慢速工艺角下f_{targ} = 250MHz、V_{DD} = 0.59V、T = 125°C），同时使用 Cadence Innovus 在相同工作点对整个集群进行布局布线。我们使用 Prime Time 对 RedMulE 进行时序分析和功耗提取，采用来自典型工艺角下 25°C 布局后仿真的 100% 标注开关活动，并针对两个工作点：470MHz、0.65V（高能效）和 613MHz、0.8V（高性能）。

5.2. 性能评估

5.2.1. 通用矩阵乘法性能评估

我们使用方形和矩形矩阵作为合成基准，评估 RedMulE 的计算延迟（以周期计），并与由 8 个共享 4 个浮点运算单元的并行 RISC-V 核心执行软件的情况进行比较。在给定基准中，RedMulE 的峰值吞吐量超过 95.4 次操作/周期，其中 ⋆和 ∘ 均计为一次“操作”，例如，对于常规通用矩阵乘法，我们将 1 次乘加运算= 2 计为 1 次操作。对于占用 55 kB 内存的96×96 FP16矩阵，RedMulE 的计算单元利用率最高可达 99.4%，在 0.80 V 电源电压下、以 613 MHz 运行时，性能达到 58.5 十亿次浮点运算每秒。图 7a 显示了在 8 个 RISC-V 核心上并行执行 FP16 软件、计算各种矩阵所需的周期数，并与 RedMulE 的结果进行比较；结果表明，在大型矩阵上，RedMulE 相较于软件执行的平均加速比达到15× 。随着矩阵规模增大，性能相较于软件执行的提升趋于 13×，因为此时软件执行也变得更加高效。我们还考察了小型 $8 \times 8 \times 8$ 情形下的加速效果，如图 7a 所示。此时加速器的利用率较低，但相较于软件并行执行，仍可实现 3.5× 的加速比。

我们通过执行一个维度固定为 M=512、N=512 和 K =512 的通用矩阵乘法内核，评估了 RedMulE 对 L、H 和 P 参数变化的敏感性。图 7b 展示了参数变化的影响。每个 RedMulE计算块包含 L 行和 $H \times (P + 1)$ 列。因此，RedMulE 每次计算访问 **X** –矩阵 的次数为 $_{M\,L} \times {}_{N\,H\times(P+1)}$，而访问 **W**-矩阵的次数为 $N \times {}_{K\,H\times(P+1)}$。考察加速器在扫描$L$ 参数（即 RedMulE 数据通路中的行数）时的行为，并考虑到矩阵的每个计算块都取决于比率 $_{M\,L}$（矩阵行数除以加速器中的行数），

X-矩阵的性能会线性提升，直到分块比率开始趋于平缓。此时，每当L 值增加能够减少**X**-矩阵上的迭代次数时，RedMulE 的性能就会阶梯式提升；随后性能再次趋于平缓，直到下一次提升。

对H 参数进行扫描时，也有类似的情况。由于 RedMulE 对**X**矩阵和**W**矩阵的访问均取决于$H \times (P + 1)$，随着H 增大，RedMulE 的性能也会提升，直至比值$_{N\,H\times(P+1)}$ 和$_{K\,H\times(P+1)}$ 趋于平稳。随后，每当增大H 参数使**X**矩阵和**W**矩阵的访问次数减少时，性能曲线便会阶跃式上升，之后再次趋于平稳。由于L的值上限为$H \times P$，扫描初始阶段L 的值也会随$H \times P$相应变化，这解释了曲线最初呈现的非线性行为。

与前述情况相比，扫描P时，性能趋势会发生变化。增加流水线寄存器的数量不仅会增大内部数据块，从而减少 RedMulE 访问**X** 矩阵和**W** 矩阵的次数，还会提高加速器的内部容量。在输入矩阵的维度保持不变时，随着流水线寄存器增加，RedMulE 的性能会下降，因为可用于馈入内部流水线队列的数据逐渐减少。因此，余数和计算时间会增加；同时，由于流水线寄存器引入了更高的延迟，性能也会下降。当P 的取值形成的数据块能够减少 RedMulE 访问矩阵的次数时，性能会再次提升；随后，由于性能与P成反比，性能会呈抛物线式下降。

5.2.2. FP16 网络训练

为进一步在实际神经网络训练案例中评估 RedMulE 的性能，我们选择了 TinyMLPerf [51], , 具体聚焦于 ResNet [52]示例。在软件基础设施方面，我们依托 PULP-TrainLib [26], , 并将 RedMulE 与运行在共享 4 个浮点运算单元的 8 个 RISC-V 核心上的软件基线进行了比较。该库考虑了梯度计算和反向传播所需的全部训练步骤。图 8a 展示了使用 8 个 RISC-V 核心并行执行时，以及使用 RedMulE 执行矩阵乘法时，ResNet8 网络单个步骤的执行情况。

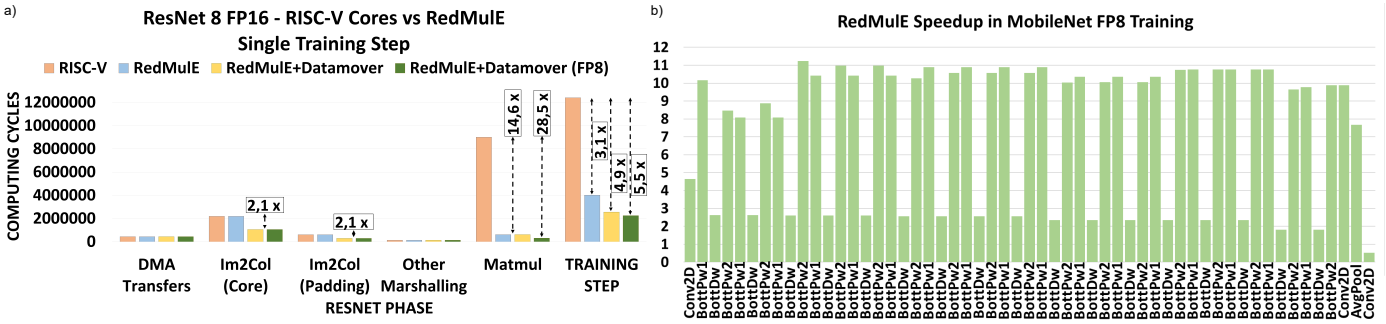


图8：RedMulE 在神经网络上的性能与由 8 个 RISC-V 核心执行的软件相比：a）ResNet8 和 b）MobileNet V2 的训练执行。

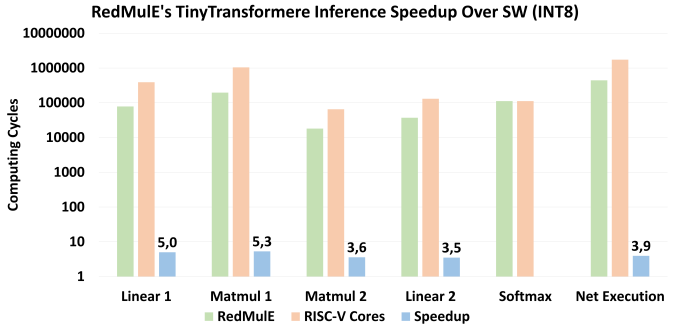


图9：RedMulE在小型 Transformer FP8 推理上的性能，与在8个RISC-V核心上运行的并行8位整数软件进行比较。

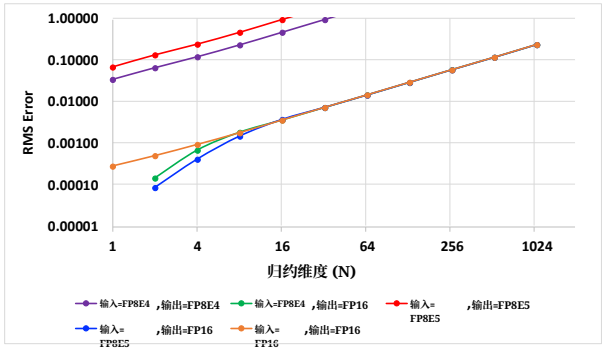


图10：不同输入/输出浮点格式在不同归约N大小下的均方根误差分析。

在并行执行以及使用 RedMulE 执行矩阵乘法时，RedMulE 将矩阵乘法执行性能提升了 14.6×，相较于软件中并行 RISC-V 执行的结果，并使 ResNet8 的整个单步执行速度提升了 3.1×。RedMulE 的利用率始终保持在 99.1%（47.6 次乘加/周期），只有在第一层和最后一层出现例外，利用率分别降至 93.2%（44.7 次乘加/周期）和 32.3%（15.5 次乘加/周期），这是由于余数无法充分利用数组的全部潜力。从图 8a 中还可以明显看出，Im2Col 期间的数据重组约占 300 万个计算周期。为解决这一问题，我们通过支持 DataMover 引擎扩展了 RedMulE 的操作，将执行两次 Im2Col 操作所需的计算周期数减半，从而使整个训练步骤的执行速度最高提升至 4.9×。由于 PULP 集群中的所有设备（RISC-V 核心和加速器）均针对协同工作进行设计并共享存储器，因此可以高效、充分地利用该架构的异构性。

5.2.3. HFP8 网络训练

对于同一个训练样例，我们考虑一个 RedMulE_{12x8}实例，仅使用 8 位 FP 输入训练 ResNet 网络。对于我们迄今考虑的 RedMulE_{12x4}，*Streamer* 的内存端口宽度为 288 位，这意味着它采用 256 位内存端口，并具备非字对齐的内存访问能力。在此配置下，RedMulE_{12x4} 每次可加载 $16 \times FP16$ 个元素，用于在计算期间填充流水线。设有 $H = 4$ 列，则每行中的流水线级数计算为 $H\times(P+1)$，其中在本实现中 $P = 3$，因此共有 16 个流水线级。对于固定的 8 位输入编码，使用相同的 288 位内存端口，RedMulE 每次最多可访问 $32 \times FP8$ 个元素。这意味着我们可以实现一个 RedMule_{12x8} 实例，在保持 288 位内存接口的同时获得 32 个流水线级。图 8a 的绿色柱形展示了 ResNet8 训练如何受益于此配置。矩阵乘法执行速度最多可提升至 28.5×，使整个训练步骤执行过程的加速比达到 5.5×，利用率为 97%。

除了 ResNet 示例外，图8b 展示了我们对 MobileNet V2 [53] 使用 FP8格式训练过程中，RedMulE 相较于并行软件所带来加速比的评估。与在配备私有浮点运算单元的8个 RISC-V 核心上经过精心优化的软件执行相比，RedMulE 平均加速比为7.5×，峰值加速比为11.2×。值得注意的是，尽管重塑后的矩阵尺寸不利，导致加速器利用率较低，RedMulE 仍能将深度卷积层的执行速度提升至多2.6×。与通用处理器核上的并行执行相比，RedMulE 将整个训练步骤加速了6.4×。

*5.2.4. FP8*变换器推理

我们评估了 RedMulE 在使用 FP8 对 TinyTransformer [54] 进行网络推理时的性能，因为 FP8 作为一种数据格式更适用于 Transformer模型 [55]。图9展示了该网络推理的结果，凸显了——

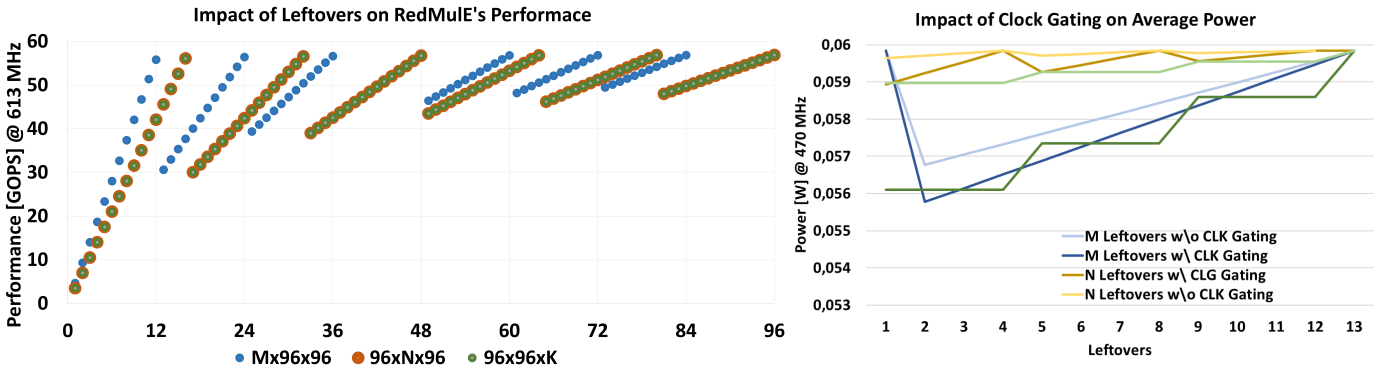


图11：细粒度时钟门控下，余数对 RedMulE 性能和能效的影响。

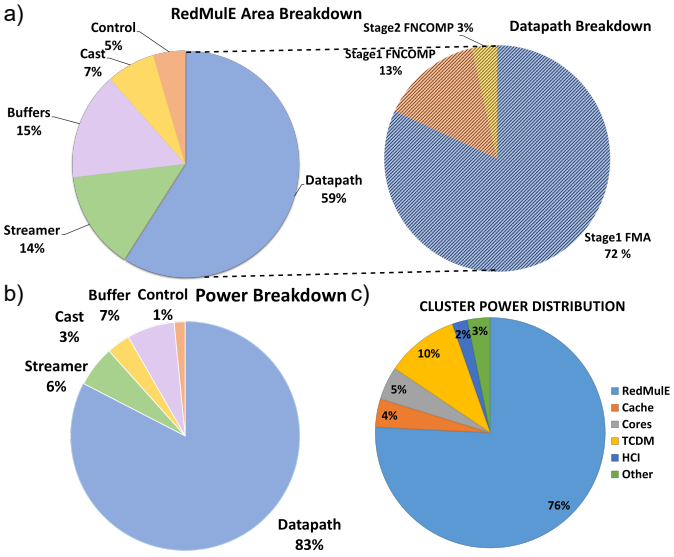


图 12：a) RedMulE 面积细分，重点关注数据通路；b) RedMulE 功耗分布；c) PULP 功耗分布。

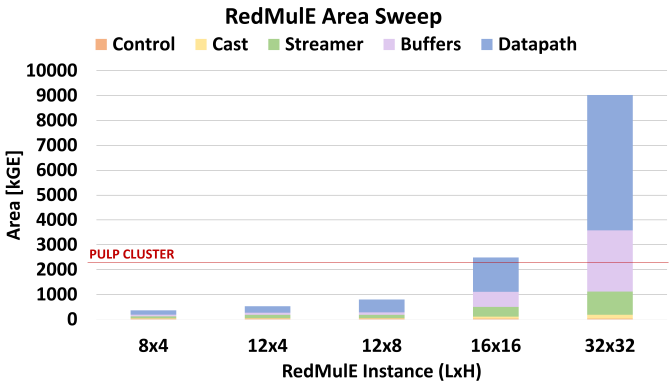


图 13：采用多种 H 和L尺寸的 RedMulE 面积扫描。

与在集群的通用处理器上并行执行相比，执行线性层和矩阵乘法层都能显著受益于 RedMulE。尽管通用处理器核通过采用 8 位整数算术的单指令多数据扩展提升了执行性能，RedMulE 在所有网络层上仍实现了超过 4× 的平均加速比；其中，Matmul1层的加速比最高，达到 5.3× ，整个网络执行的加速比则为 3.9× 。

5.3. 输入和输出张量格式的误差分析

我们针对**X** 和**W** 矩阵的归约维度（N）逐渐增大的情况进行了误差分析。CE中的FMA单元采用固定的内部精度，从而减少中间累加步骤中的精度损失。然而，当**X**、**W**和**Z** 均采用8位精度表示时，与16位输入、16位输出的情况相比，均方根误差（RMSE）增加超过 100× ，且这一趋势与N 的大小呈指数关系。不过，仅将**X** 和**W** 采用8位精度表示，同时令输出**Z**矩阵保持较宽的16位精度，与仅采用16位精度的情况相比，精度损失可以忽略不计。

5.4. RedMulE 面积

5.4.1. 面积细分分析

RedMulE_{12x4} 占用0.15mm^2，，相当于整个PULP集群面积（0.64mm^2）的23.8%。图12a展示了RedMulE 面积的细分情况，其中类型转换单元占加速器总面积的7%，计算单元占59%，而计算单元面积中有72%来自FMA单元。

*5.4.2. RedMulE*面积扫描

我们研究了改变 RedMulE 内部计算单元数量所带来的面积开销，并将计算单元的内部流水线级固定为 $P = 3$。图13显示，当 RedMulE 包含256个计算单元时，其面积占用与整个 PULP 集群的面积相当，对应于一个 RedMulE_{16x16} 实例。另一方面，RedMulE_{32x32} 的面积比整个 PULP 集群大 4× 。图13还显示，改变数据通路 的形状也会影响*Streamer*的尺寸。具体而言，每当数据通路的一行中增加一个计算单元（或等效地，改变H 参数），每个数据通路 行中就会增加其他 $P + 1$ 个流水线寄存器。因此，为保持较高的计算单元利用率而需要的元素数量也会增加$P+1$ 。以$P = 3$ 为例，将H 参数增加1，就要求 *Streamer* 向数据通路提供额外的$P + 1$（$= 4$）个 FP16元素。因此，流处理器端口必须扩宽64位（$= 4 \times 16$–位），从而限制了 RedMulE 集成到 PULP 集群中的能力。

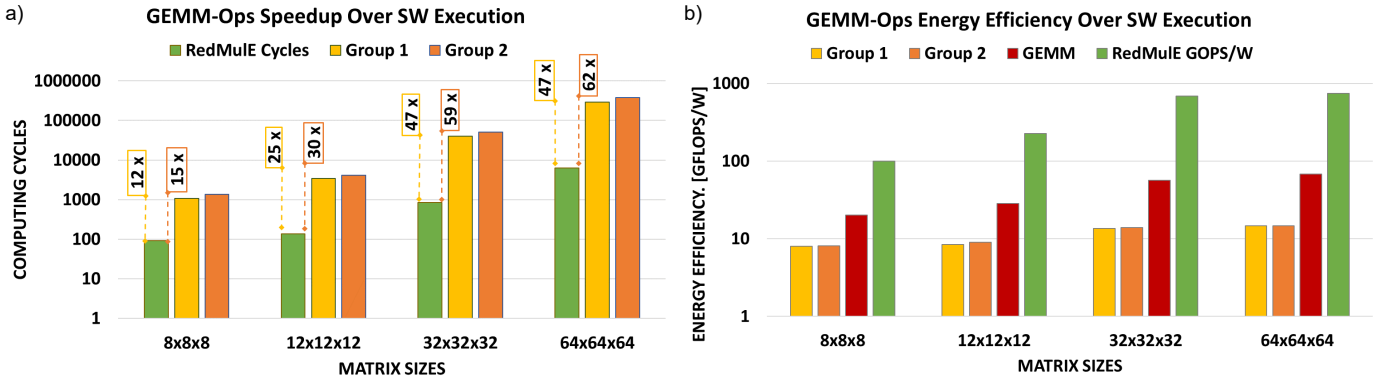


图14：RedMulE 在执行通用矩阵乘法运算时相较于并行软件实现的 a）性能和 b）能效。

*5.5. RedMulE*功耗

在集群层面，效率工作点下通用矩阵乘法操作期间的功耗为59.3mW。RedMulE 的功耗占主导地位，占总功耗的66.8%；紧耦合数据存储器存储体和 HCI互连的功耗则占13.3%。在这一工作点，通用矩阵乘法执行期间集群的峰值能效达到755GFLOPS/W，相较于软件基线，能效提高了12.5× 。图12b和图12c分别展示了 RedMulE 的功耗分解（其中大部分功耗由数据通路消耗），以及通用矩阵乘法操作期间 PULP 集群的功耗分解（其中大部分功耗由 RedMulE 和紧耦合数据存储器存储体消耗）。

5.6. 时钟门控评估

我们评估了余数对 RedMulE 性能和能效的影响，以及细粒度时钟门控如何帮助降低利用率不足阶段的功耗。图11展示了我们的分析结果：为评估性能损失，我们每次改变输入矩阵的一个维度（M、N或K），并保持另外两个维度不变，同时将 RedMulE 的参数固定为L=12、H=4和P=3。为评估对功耗的影响，我们分析了最后一个性能步对应时间窗口内 RedMulE 启用和未启用时钟门控贡献时的功耗。

在图11中，性能图中的蓝点突出显示了改变**X**矩阵行数（M）时的性能趋势。**X**矩阵的行中存在余数，意味着 RedMulE 只使用一定数量的行（从一行到L行）来计算矩阵中的相应部分。因此，RedMulE阵列的一整部分不会执行任何计算。此外，由于 RedMulE 只需要使用X 和Z 缓冲区的一部分，这些缓冲区的活动也会降低。从性能角度来看，图11显示，当M 从 1 增至 12 时，性能从 4.7 十亿次运算每秒提升至 55.8 十亿次运算每秒。在这种情况下，**X**矩阵只有一个分块与所有**W**矩阵分块相乘。因此，余数会影响整个矩阵乘矩阵运算。随着M增大，数组得到充分利用的计算周期数逐渐增加，从而平衡了余数所造成的性能损失。

图11右侧展示了在计算余数期间，对数组中未使用的计算单元以及X 和Z 缓冲区中未使用部分进行时钟门控所产生的功耗趋势。当**X**矩阵的行数（M）远小于**X** 矩阵的列数（N）时，所提出的时钟门控方案最多可节省加速器 22% 的功耗。在这种情况下，计算元件阵列中有很大一部分未得到充分利用，因此在大部分计算期间都处于时钟门控状态。例如，执行 MobileNet 的瓶颈深度卷积层时就会出现这种情况，这些层可以重塑为向量矩阵乘法（$M = 1, N >> 1$）。

改变N 和K 产生的影响相似。N 和K 维度上的余数会导致每个加速器行中的流水线寄存器利用率较低。从性能角度看，这与M 变化时的情况类似。不过，曲线斜率较低，因为在H=4 和P=3的配置下，每行有16个（$= H \times (P + 1)$）内部流水线寄存器；每当N 或K 的值达到$H \times (P + 1)$的整数倍时，每条绘图曲线的性能就会接近最大值。与M可变时的情况类似，在此情况下，第一条曲线也从最低性能值 3.5十亿次运算每秒开始，因为余数会影响整个计算过程。从第二条曲线到最后一条曲线，RedMulE 完全利用 CE阵列进行计算的周期，能够弥补矩阵余数带来的影响。从功耗角度看，由于 RedMulE 在每个 CE 列之间设有四个（$= P + 1$）流水线寄存器，绘图曲线每经过四个矩阵元素就会阶跃一次。所提出的时钟门控方法可以关闭余数计算期间未激活的 CE 列，因此计算余数时，激活的计算单元数量可从一个（功耗曲线的第一步）到四个（每条功耗曲线的最后一步）不等。所提出的时钟门控方案在严重利用率不足的情况下可节省高达37%的平均功耗，因此与未采用时钟门控的情况相比，能效提高了60%。

*5.7. GEMM-Ops*测量

为评估通用矩阵乘法运算性能，我们在图14a中将 RedMulE 的通用矩阵乘法运算执行与 RISC-V 核心上的并行软件执行进行了比较。RedMulE 执行任何受支持的通用矩阵乘法运算时，始终需要相同数量的计算周期。相比之下，

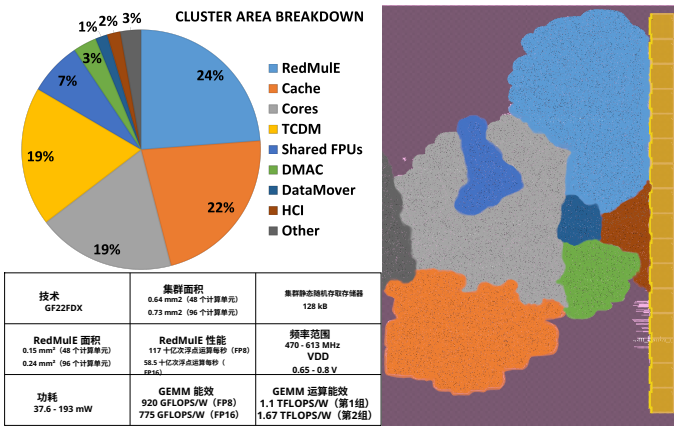


图 15：PULP集群的面积分解、布局和摘要表。

通用处理器核上的并行执行所需时间会根据所执行的核函数而变化。属于第1组的所有核函数（见表1），即其中 ∘ 操作为 +/x 且 ⋆ 操作为 max/min 的核函数，在核心上执行时所需的计算周期数相同。在 RedMulE 上执行这些核函数时，加速比最高可达 47× 。当 ⋆ 也属于 max/min 类型，即第2组时，通用处理器核的执行开销更高，而 RedMulE 可将此类核函数的执行速度提高至 62×。

图 12a 显示，支持通用矩阵乘法运算仅使整个加速器面积增加了 16%。其中 13% 的开销来自第一级 FNCOMP单元，主要是为了匹配 FMA单元的延迟周期而引入的流水线所致。第二级 FNCOMP单元完全由组合逻辑构成，开销仅占 3%。

我们评估了 RedMulE 在执行通用矩阵乘法运算期间的功耗和能效。图 14b 比较了 RedMulE 与软件基线的能效；该软件基线由 8 个 RISC-V 核心和 4 个共享浮点运算单元组成，在执行通用矩阵乘法、通用矩阵乘法运算第1组和第2组核函数时处理 FP16元素。对于通用矩阵乘法运算第1组，集群级功耗达到 53.2 mW，能效为 842 GFLOPS/W，比通用处理器核上的并行软件执行高 57.2×。另一方面，执行通用矩阵乘法运算第2组中的算法时，功耗进一步降至 37.6 mW，能效达到 1.19 TFLOPS/W，较并行软件实现提高 81.2×。

6. 与最先进水平的比较

表 2 总结了我们与不同最先进水平（SoA）架构的比较。

我们将本工作与 GPU架构进行比较，尤其是包含 Tensor Cores 的 NVIDIA H100。其 FP16 性能最高可达 989TFLOPS，FP8 性能最高可达 1979TFLOPS，这意味着 17000× 比我们的工作更高，但代价是 700W 的功耗和 814mm^2， 12000× 比我们的工作功耗更高、 1300× 体积更大——对于物联网终端节点而言，这是一种难以实际应用的解决方案。

尽管 RedMulE 主要面向训练，但也可用于推理。因此，我们在比较中纳入了一些面向推理的芯片，例如 DNPU [21]。DNPU 的性能仅比我们的集群高 1.9× ，尽管 DNPU 的计算单元数量是我们的 16× 倍。此外，DNPU 的能效比 RedMulE 高 2.7× ，但它仅支持定点精度，这有助于提高能效。我们还将本工作与 Diana [20] 和 Gemmini [22], 进行了比较，它们与 RedMulE 采用相同的工艺节点设计。前者的性能比 RedMulE 低 44.5% $_{12x8}$ ，在能效模式下的性能比 RedMulE 低 12%$_{12x4}$ 。Diana 在能效模式下的功耗远低于我们的设计，但如果像他们那样将频率降至 50 MHz，我们搭载 RedMulE 的 PULP 集群$_{12x4}$ 仅消耗 7.65 mW。Diana 的能效大幅提升是有原因的：它仅采用 8位整数格式，这有助于降低功耗并提高能效。相比之下，尽管 Gemmini 的计算单元数量是 5×，且采用 8位整数格式，其能效仍比 RedMulE 低一个数量级$_{12x4}$ 。

我们还将设计与其他专为片上训练设计的平台进行了比较。IBM [35] 展示了一款采用 7 nm 工艺的 4 核人工智能芯片，尽管工艺缩放且工作电压降低，其能效仅比搭载 RedMulE$_{12x4}$的 PULP 集群高 2.4× ，面积大 33.2× ，功耗高 74× 。IBM 还提出了一款功耗超过 1 W 的芯片 [37], ，这无法满足极端边缘训练的要求。另一方面，LNPU [38] 是一款极端边缘处理器，其功耗范围比 RedMulE$_{12x4}$高 6.67× 。Vega 是片上嵌入式训练的可行候选方案，但 RedMulE$_{12x4}$的性能高 7.8× ，能效高 3.2× 。寒武纪-Q [30] 采用 45 nm 工艺设计，能效比我们的设计高 2.9×，但使用较低精度的 8位定点运算；而通常可用的、基于反向传播的学习算法严格要求浮点范围和精度。寒武纪-Q 的功耗也比我们的设计高 17.7× ，因此不适用于 TinyML 应用。类似的考量也适用于 T-PIM [39], ： 这款采用 28 nm 工艺设计的训练芯片配备了存内计算核心，能效较高，但仅支持 16 位整数精度，无法满足实现片上训练所需的精度要求。

TSUNAMI [41] 和 Trainer [42]旨在实现高能效的嵌入式训练，并广泛采用剪枝和稀疏矩阵生成，以提升能效，并通过零跳过减少训练所需的乘加运算次数。我们将其与它们在密集计算中的结果进行比较。在最佳效率点，TSUNAMI 和 Trainer 的功耗与 RedMulE 相当。然而，这些效率点对应的频率分别为 TSUNAMI 的 50 MHz 和 Trainer 的 40 MHz，而 RedMulE 的评估频率为 470 MHz。因此，在相近频率下，RedMulE 的功耗约低一个数量级。TSUNAMI 和 Trainer 的性能最高分别可提升 5× 和 8× ，因为它们采用了 21× 和 85×

表 2：技术现状对比。第一行 = 最佳效率；第二行 =峰值性能。1 次乘加运算 = 2 操作次数。

类别	设计	Tech *nm*	Area mm^2	Freq *MHz*	Volt *V*	功耗 *mW*	Perf *GOPS*	能量 Eff 十亿次运算每秒/*W*	CEs	精度
GPU	英伟达 H100 [15]	4	814	1830	-	700000	1978900 989400	2827 1413	528	FP8 FP16
	SIMD^2 [6]	45	19.5	-	-	4190	-	-	-	16位整数
推理芯片	DNPU [21]	65	16	50 200	0.7 1.1	34.6 279	72.6 279	2100 1000	768	16位整数
	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
训练芯片	4核 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	-	16位整数
	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
	寒武纪-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
矩阵乘法	Anders [43]	14	0.024	2.1 1090	0.26 0.9	0.023 82.7	0.068 34	2970 420	16	FP16
GEMM	**本工作 RedMulE$_{12x4}$**	**22**	**0.64**	**470 613**	**0.65 0.8**	**59.3 116**	**44.8 58.5**	**775 506**	**48**	**FP16**
第1组 通用矩阵乘法运算						**53.2 103**		**842 576**		
第2组 通用矩阵乘法运算						**37.6 71.5**		**1193 819**		
GEMM	**本工作 RedMulE$_{12x8}$**	**22**	**0.73**	**470 613**	**0.65 0.8**	**97.5 193**	**89.7 117**	**920 608**	**96**	**FP8**
第1组 通用矩阵乘法运算						**85.2 168**		**1052 694**		
第2组 通用矩阵乘法运算						**54 104**		**1666 1123**		

计算单元数量相同，但计算单元利用率远低于我们的 RedMulE（TSUNAMI 为 75%，Trainer 仅为 12.5%）。RedMulE 的脉动架构原则上支持几乎任意规模的架构扩展。假设性能、面积和功耗均按比例线性扩展，将规模扩展至 1024 或 4096 个计算单元（ 21× 且 85× 比 RedMulE$_{12x4}$ 更大），我们的利用率仍可达到 99.4%；计入开销后，总体性能将分别达到 1.25 TFLOPS 和 5 TFLOPS。

我们将 RedMulE$_{12x4}$ 与 Anders 等人 [43], 提出的硬件加速器进行比较。该加速器采用 14 纳米工艺，面向 TinyML 学习和推理应用中的矩阵乘法。其 FP16 精度下的峰值能效达到 2.97 TFLOPS/W，比 RedMulE$_{12x4}$高 3.83× ，但这只有在近阈值电压（260 mV）和极低频率（2.1 MHz）下才能实现。在该工作点，其设计的性能比 RedMulE 低 659×。Anders 的峰值性能是在 0.9 V 和 1.09 GHz 下实现的，能效为 420 GFLOPS/W，乘累加单元利用率为 99.4%，与 RedMulE 类似。在 22 纳米工艺下，当频率为 613 MHz、电压为 0.8 V 时，RedMulE$_{12x4}$ 达到 58.5 GFLOPS，比 Anders 等人高 1.72× ； 其 FP16 通用矩阵乘法能效也高出 20.5%，达到 506 GFLOPS/W。

我们还将 RedMulE 与 SIMD^2 [6], 进行了比较，后者是唯一采用通用矩阵乘法操作扩展的设计。尽管 SIMD^2 仅支持整数算术，因此无法用于片上训练，但其功耗比 RedMulE 高 36.1× 。在他们的研究中，作者还指出，在 NVIDIA 流式多处理器上构建通用矩阵乘法操作扩展所带来的面积开销为 69%；而在 RedMulE 中，通用矩阵乘法操作扩展引入的面积开销仅为 16%。

7. 结论

本文介绍了 RedMulE——低精度矩阵乘法引擎，这是一款全参数化的开源集群耦合加速器，可在超低功耗设备上实现 TinyML 训练，即在仅有几十毫瓦功耗预算的条件下进行近传感器训练。RedMulE 专为 FP16 通用矩阵乘法运算而设计，也支持压缩 FP8 输入，同时还能高效加速大量具有与通用矩阵乘法相同结构的运算。RedMulE 支持灵活实例化各种基于浮点单元的计算单元（CE）、内部缓冲区和存储器接口配置。我们将一个包含 12×4 CE 阵列的 RedMulE 实例集成到一个配备 8 个 RISC-V 核心的超低功耗集群中，并采用 22 纳米工艺实现了整个系统。RedMulE 的计算单元利用率达到 99.4%；在简单通用矩阵乘法执行期间，相较于在八个核心上运行的并行软件基线，其平均加速比为 15× 。RedMulE 占用 0.15 mm^2 ，仅占集群面积的 24%。在执行 GEMM 操作时，RedMulE 相较于 RISC-V 核心带来的性能加速比最高可达 62×。在其最佳性能工作点（613 MHz、0.8 V），执行浮点通用矩阵乘法内核时，RedMulE 的性能达到 58.5 GFLOPS，能效达到 506 GFLOPS/W；在其最佳能效工作点（470 MHz、0.65 V），能效则达到 775 GFLOPS/W

，性能达到 44.8 GFLOPS。在一个实际的神经网络训练示例中，当输入张量分别采用 16 位和 8 位表示时，RedMulE 可将矩阵乘法加速至最高 14.6× 倍和 28.5× 倍，使 ResNet8 的整个训练步骤分别加速 4.9× 倍和 5.2× 倍。

致谢

本工作部分得到了 Thales AleniaSpace、欧洲 PILOT 项目（欧洲高性能计算联合事业，拨款协议号 101034126）以及 Neu-roSoC（欧盟地平线计划，拨款协议号 101070634）的支持。本工作还部分得到了“TinyTrainer”项目的支持，该项目由瑞士国家科学基金会资助（拨款协议号 20791）。

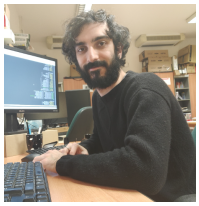
参考文献

[1] M. Satyanarayanan，《边缘计算的兴起》，《计算机》（第50卷，第1期，2017年1月）（2017年）。[2] J. Gilbert、S. Reinhardt、V. B. Shah，《数值与组合计算的统一框架》，《科学与工程计算》（第10卷，第2期，2008年3月至4月）（2008年）。[3] J. Gilbert、S. Reinhardt、V. B. Shah，《基于并行稀疏矩阵的高性能图算法》，《应用并行计算：科学计算技术现状》。PARA 2006（2008年）。[4] M. Mohri，《最短距离问题的半环框架与算法》，《自动机、语言与组合学杂志》7（3）（2002年）321–350。[5] S. G. Sedukhin、M. Paprzycki，《为现代计算机上的高效计算推广矩阵乘法》，载于：R. Wyrzykowski、J. Dongarra、K. Karczewski、J. Wa´sniewski（编），《并行处理与应用数学》，施普林格·柏林·海德堡，柏林，海德堡，2012年，第225–234页。[6] Y. Zhang、P. Tsai、H. W. Tseng，《SIMD2：用于加速通用矩阵乘法之外张量计算的通用矩阵指令集》，ISCA ’22：第49届年度计算机体系结构国际研讨会论文集（2022年）。[7] L. Ravaglia、M. Rusci、A. Capotondi、F. Conti，《RISC-V极端边缘节点上持续学习的存储器-延迟-精度权衡》，IEEE 信号处理系统研讨会（SiPS）（2020年）。[8] 英伟达，《混合精度训练——英伟达深度学习性能文档》， https://tinyurl.com/2p944wfs （2021年）。[9] A. F. Rodriguez Perez、B. Ziv、E. M. Fomenko、E. Meiri、H. Shen，《低数值精度的深度学习推理与训练》， 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，《深度学习的FP8格式》，ArXiv预印本 arXiv:2209.05433（2022年）。[11] X. Sun、J. Choi、C. Chen、N. Wang、S. Venkataramani，《用于深度神经网络的混合8位浮点（HFP8）训练与推理》，IBM 托马斯·J·沃森研究中心，纽约州约克镇高地，10598，美国（2019年）。[12] G. Tagliavini、S. Mach、D. Rossi、A. Marongiu、L. Benini，《一种用于超低功耗计算的可变精度浮点平台》，载于：2018年欧洲设计、自动化与测试会议暨展览会（DATE），2018年，第1051–1056页。 doi:10.23919/DATE.2018.8342167。[13] L. Bertaccini、G. Paulin、T. Fischer、S. Mach、L. Benini，《Minifloat-nn与exsdotp：用于RISC-V核心上低精度训练的指令集架构扩展和模块化开放硬件单元》，载于：2022年IEEE第29届计算机算术研讨会（ARITH），美国电气与电子工程师学会计算机学会，洛斯阿拉米托斯，加利福尼亚州，美国，2022年，第1–8页。 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，《人工智能与机器学习加速器综述及趋势》，载于：2022年IEEE高性能极端计算会议（HPEC），2022年，第1–10页。doi:10.1109/HPEC55821.2022.9926331。

[15] 英伟达，Nvidia h 100 https:// 张量核心 resources.nvidia.com/en-us-tensor-core 2022 GPU架构，（）。[16] Z. Zhou、X. Chen、E. Li、L. Zeng、K. Luo、J. Zhang，《边缘智能：通过边缘计算铺就人工智能的最后一公里》，《并行处理与应用数学》（2019年）。[17] A. N. Mazumder、J. Meng、H.-A. Rashid、U. Kallakuri、X. Zhang、J.-S. Seo、T. Mohsenin，《面向微型人工智能设备端推理的神经网络加速器优化综述》，IEEE 新兴与精选电路和系统主题期刊 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，《面向异构物联网边缘设备的自感知分布式深度学习框架》，《未来一代计算机系统》125（2021年）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，《低功耗人工智能加速器的最新进展：综述》，《算法》 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：面向边缘的端到端混合数字与模拟神经网络片上系统》，IEEE固态电路期刊58（1）（2023年）203–215。 doi:10.1109/JSSC.2022.3214064。[21] D. Shin、J. Lee、J. Lee、J. Lee、H.-J. Yoo，《DNPU：一种采用异构多核架构的节能深度学习处理器》，《IEEE微型计算机》38（5）（2018年）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. Asanovi´c、B. Nikoli´c，《一款采用低功耗22nm FinFET工艺、面积为16mm2、能效达106.1 GOPS/W的异构RISC-V多核多加速器片上系统》，载于：ESSCIRC 2021——IEEE 第47届欧洲固态电路会议（ESSCIRC），2021年，第259–262页。 doi:10.1109/ESSCIRC53450.2021.9567768。[23] C. Frenkel、M. Lefebvre、D. Bol，《无反馈学习：固定随机学习信号支持深度神经网络的前馈训练》，《神经科学前沿》15（2021年）。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：在微控制器上进行在线学习的TinyML》，ArXiv预印本 arXiv:2103.08295（2021年）。[25] V. Ramanathan，《在线设备端微控制器迁移学习》（2020年）。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：通过性能驱动的自动调优，支持在RISC-V多核微控制器上进行设备端训练》，载于：A. Orailoglu、M. Reichenbach、M. Jung（编），《嵌入式计算机系统：架构、建模与仿真》，Springer 国际出版公司，Cham，2022年，第200–216页。[27] C. Cioflan、L. Cavigelli、M. Rusci、M. De Prado、L. Benini，《迈向抗噪关键词识别的设备端领域自适应》，载于：2022年IEEE第四届人工智能电路与系统国际会议（AICAS），2022，82 85 doi:10.1109/AICAS54282. 2022.9869990 页。[28] L. Ravaglia、M. Rusci、D. Nadalini、A. Capotondi、F. Conti、L. Benini，《支持量化潜在重放的设备端持续学习TinyML平台》，IEEE 新兴与精选电路和系统主题期刊11（4）（2021年）789–802。 doi:10.1109/JETCAS.2021.3121554。[29] D. Rossi等，《Vega：一款面向物联网终端节点、具备深度神经网络加速和基于磁阻式随机存取存储器的状态保持睡眠模式认知唤醒功能的十核片上系统》，IEEE固态电路期刊57（1）（2021年）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，《寒武纪-Q：一种用于高效训练的混合架构》，载于：2021年美国计算机协会/电气与电子工程师学会第48届计算机体系结构国际研讨会（ISCA），2021年，第706–719页。 doi:10.1109/ISCA52012.2021.00061。[31] B. Noune、P. Jones、D. Justus、D. Masters、C. Luschi，《深度神经网络的8位数值格式》，ArXiv预印本 arXiv:2206.02915（2022年）。 [32] J. Park, S. Lee, D. Jeon，《一种采用 8 位共享指数偏置浮点和多路融合乘加树的神经网络训练处理器》，IEEE固态电路期刊 57 (3) (2022) 965–977。 doi:10.1109/JSSC.2021.3103603.[33] S. Fox, S. Rasoulinezhad, J. Faraone, david boland, P. Leong，《用于训练深度神经网络的分块微浮点表示》，载于：国际学习表征会议，2021。URL https://openreview.net/forum?id=6zaTwpNSsQ2 [34] N. Wang, J. Choi, D. Brand, C.-Y. Chen, K. Gopalakrishnan，《使用 8位浮点数训练深度神经网络》，载于：第32届神经信息处理系统国际会议论文集，NIPS'18，Curran Associates Inc.，美国纽约州雷德胡克，2018年，第7686–7695页。 [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，《9.1：一款采用 7纳米四核AI芯片的芯片，支持 25.6 TFLOPS 混合FP8训练、102.4 TOPS INT4推理和工作负载感知节流》，载于：2021年 IEEE 国际固态电路会议（ISSCC），第64卷，2021年，第144–146页。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：用于超低精度训练与推理的AI加速器》，载于：2021年美国计算机协会/电气与电子工程师学会第48届计算机体系结构年度国际研讨会， 2021， 153 166 doi: 第–页。[37] 10.1109/ISCA52012.2021.00021 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，《一款用于高计算利用率 AI 训练与推理的 3.0 TFLOPS、0.62 V 可扩展处理器核心》，载于：2020年 IEEE超大规模集成电路研讨会， 2020， 1 2 doi:10.1109/VLSICircuits18222. 第–页。[38] 2020.9162917 J. Lee, J. Lee, D. Han, J. Lee, G. Park, H.-J. Yoo，《7.7 LNPU：一款支持FP8-FP16 细粒度混合精度的 25.3 TFLOPS/W 稀疏深度神经网络学习处理器》，载于：2019年 IEEE 国际固态电路会议（ISSCC）， 2019， 142 144 doi:10.1109/ISSCC. 第–页。[39] 2019.8662302 J. Heo, J. Kim, S. Lim, W. Han, J.-Y. Kim，《T-PIM：一种用于端到端设备端训练的高能效存内处理加速器》，IEEE固态电路期刊 58 3 2023 600 613 doi:()()–。[40] 10.1109/JSSC.2022.3220195 H. Bian, J. Huang, L. Liu, D. Huang, X. Wang，《ALBUS：一种使用 SIMD 和负载均衡高效处理 SPMV 的方法》，《未来一代计算机系统》 116 2021 371 392 doi:https://doi.org/()–。URL 10.1016/j.future.2020.10.036 https://www.sciencedirect.com/science/article/pii/ S0167739X2033020X[41] S. Kim, J. Lee, S. Kang, D. Han, W. Jo, H.-J. Yoo，《TSUNAMI：采用多模态迭代剪枝的三重稀疏性感知超高能效神经网络训练加速器》，IEEE电路与系统汇刊第一辑：正规论文 69 4 2022 1494 1506 doi:10.1109/()()–。[42] TCSI.2021.3138092 Y. Wang, Y. Qin, D. Deng, J. Wei, T. Chen, X. Lin, L. Liu, S. Wei, S. Yin，《Trainer：支持动态权重剪枝的高能效边缘设备训练处理器》，IEEE固态电路期刊 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, 《一款采用统一 int8/inti6/fp16 数据通路、基于 14 纳米三栅极 CMOS 的 2.9 TOPS/W 可重构稠密/稀疏矩阵乘法加速器》，载于：2018 年 IEEE 超大规模集成电路研讨会，2018年，第39–40页。doi:10.1109/VLSIC.2018.8502333.[44] G. Mehrooz, P. Schneider-Kamp, 《线性基础设施无人机巡检的最优路径规划》，载于：国际地理信息系统理论、应用与管理会议，2020年。[45] S. Suk, Minsoo ; Sull, 《曲线特征提取与近似》，《计算机视觉、图形学与图像处理》。(1983 400 411doi:)–。[46] Y. 10.1016/0734-189X(84)90221-4 Tortorella, L. Bertaccini, D. Rossi, L. Benini, F. Conti, 《RedMulE：一款用于在基于 RISC-V 的超低功耗片上系统上实现自适应深度学习的紧凑型 fp16 矩阵乘法加速器》，载于：2022 年欧洲设计、自动化与测试会议暨展览会（DATE），2022年，第1099–1102页。doi:10.23919/DATE54114.2022.9774759.[47] A. Pedram, R. A. van de Geijn, A. Gerstlauer, 《高性能、低功耗线性代数架构的协同设计权衡》，IEEE 计算机汇刊 61 (12) (2012) 1724–1736。 doi:10.1109/TC.2012.132. [48] F. Conti, P. D. Schiavone, L. Benini, 《Xnor 神经引擎：一种用于二值神经网络推理的 21.6 飞焦耳/运算硬件加速器 IP》，IEEE集成电路与系统计算机辅助设计汇刊 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：一种用于极端边缘端片上深度神经网络推理与训练的异构 RISC-V 计算集群》，IEEE 固态电路学会开放期刊 2 2022 231 243 doi:10.1109/()–。[50] OJSSCS.2022.3210082 S. Mach, F. Schuiki, F. Zaruba, L. Benini, 《FPnew：一种用于能效成比例的跨精度计算的开源多格式浮点单元架构》，IEEE 超大规模集成电路系统汇刊，第29卷，第4期，第774–78页（2020年）。[51] C. R. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kir´aly, 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 基准测试》，ArXiv abs/2106.07597 (2021)。[52] K. He, X. Zhang, S. Ren, J. Sun, 《用于图像识别的深度残差学习》（2016）770–778doi:10.1109/CVPR.2016.90。[53] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, 《MobileNetV2：倒置残差与线性瓶颈》，载于：2018 年 IEEE/CVF 计算机视觉与模式识别会议，2018年，第4510–4520页。 doi:10.1109/CVPR.2018.00474.[54] A. Burrello, M. Scherer, M. Zanghieri, F. Conti, L. Benini, 《你只需要一台微控制器：在低功耗物联网终端节点上实现 Transformer 执行》，载于：2021年 IEEE 国际全层智能系统会议（COINS），2021， 1 6 doi:10.1109/COINS51742. 第–页。[55] 2021.9524173 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 与 INT8：面向高效深度学习推理》，Arxiv 预印本 arXiv:2303.17951v1（2023）。



Yvan Tortorella 获得了硕士学位他于2021年10月获得博洛尼亚大学电子工程学位。目前，他正在博洛尼亚大学电气与信息工程系（DEI）Luca Benini教授领导的数字系统设计组攻读博士学位。他的研究兴趣包括设计基于PULP（并行超低功耗）的超低功耗机器学习硬件加速器，以及设计用于卫星应用的基于RISC-V的计算机架构。



Luca Bertaccini获得了理学硕士学位他于2020年获得博洛尼亚大学电子工程学位。目前，他正在ETH Z¨urich由Luca Benini教授领导的数字电路与系统组攻读博士学位。他的研究兴趣包括异构片上系统、高能效硬件加速器、计算机算术和跨精度计算。他获得了2021年IEEE ASAP最佳论文荣誉提名奖。



Luca Benini 担任数字电路与系统学讲席教授，任职于苏黎世联邦理工学院（ETHZ），并担任博洛尼亚大学正教授。他获得了斯坦福大学博士学位。Benini博士的研究兴趣包括高能效并行计算系统、智能传感微系统和机器学习硬件。他发表了1000多篇经同行评审的论文和五本书。他是电气与电子工程师学会和美国计算机协会会士，也是欧洲科学院成员。他于2016年获得IEEE Mac Van Valkenburg奖，并于2020年获得ACM/IEEE A. Richard Newton奖。



DavideRossi 获得博士学位于2012年毕业于意大利博洛尼亚的博洛尼亚大学。自2015年起，他一直在博洛尼亚大学“Guglielmo Marconi”电气、电子与信息工程系担任博士后研究员，目前为副教授。他的研究主要聚焦于高能效数字架构。在这一领域，他已在国际同行评审会议和期刊上发表论文100余篇。他曾获2018年Donald O. Pederson最佳论文奖、2020年IEEE TCAS Darlington最佳论文奖以及2020年IEEE TVLSI优秀论文奖。


Francesco Conti 获得博士学位2016年毕业于意大利博洛尼亚大学电子工程专业。他目前是博洛尼亚大学DEI系的终身教职轨道助理教授。2016年至2020年，他曾获得博洛尼亚大学的研究资助，并在苏黎世联邦理工学院担任博士后研究员。他的研究主要聚焦于超低功耗、高能效平台上的硬件加速，尤其关注面向人工智能应用的片上系统。他的研究成果已发表于国际会议和期刊的论文超过70篇，并多次获奖，其中包括2020年IEEE TCAS-I达灵顿最佳论文奖。