TensorFlow Lite 中动态可重构的变精度稀疏-稠密矩阵加速

Jose Nunez-Yanez、Andres Otero、Eduardo de la Torre

arXiv:2304.08211v1 [cs.AR] 17 Apr 2023

摘要——本文介绍了一种名为 FADES（稠密与稀疏矩阵融合架构）的动态可重构硬件加速器。FADES 设计提供多种配置选项，可在并行性与复杂度之间进行权衡，并采用数据流模型创建四个阶段，分别用于读取、计算、缩放和写入结果。FADES 映射到可编程逻辑（PL），并与运行在异构片上系统设备处理系统（PS）上的 TensorFlow Lite 推理引擎集成。该加速器用于执行张量运算，而动态可重构方案则可在*int8* 和浮点数 模式之间切换精度。这种动态重配置通过允许将更多处理核心映射到资源受限的设备上，提升了性能；与同时支持两种算术精度相比，还能降低功耗。我们将所提出的硬件与用于稠密矩阵的高性能脉动架构进行了比较；在采用相同技术的情况下，FADES 在稠密模式下的性能提升了 25%，且所需 DSP 模块数量减半。在稀疏模式下，即使稀疏度较低，处理核心的性能也能超过稠密模式；单核相较于经软件优化的 NEON RUY 库，最高可实现 20 倍加速。

索引词—神经网络、现场可编程门阵列、稀疏、剪枝、矩阵乘法加速、TensorFlow

1 引言

在本研究中，我们提出了 FADES（稠密与稀疏张量处理融合架构）数据流引擎，并将其扩展为具备动态重配置（即 Xilinx DFX）能力，以支持浮点和 8 位精度运算。DFX 允许通过下载部分比特文件来修改逻辑块，从而在不打断系统其余部分运行的情况下实时改变其功能。DFX 能够更高效地利用硅片，使设计人员可以选用更小的设备并降低功耗。FADES 集成为 TensorFlow Lite（TFLite）的加速器；TensorFlow Lite 是谷歌面向移动设备和嵌入式设备的轻量级方案，适用于其倡导的边缘部署。通过这种集成，FADES 得以受益于 TensorFlow Lite 社区在先进的量化感知训练和剪枝方面开展的大量前期研究 [1]，并直接取代高性能矩阵乘法库 RUY [1]。能够同时执行浮点和 8 位运算的硬件加速器很有用，因为 TFLite 原生支持这两种精度数据类型。深度学习模型不断演进，因此需要灵活的硬件方案，以适应日益增长的网络复杂度和多样性。基于这些观察，本文的贡献如下：

- 我们介绍采用高层次综合方法设计的 FADES 可变精度稀疏-稠密数据流加速器，以及将其集成到 TensorFlow Lite 推理引擎中。
- 我们开发了适用于稀疏和稠密矩阵处理模式的8位整数及浮点配置，并将其性能与脉动阵列硬件加速器和优化软件方案进行了比较。
- 我们提出了一种结合高级综合与动态功能交换的设计方法，用于生成并部署可变精度神经网络加速器。
- 我们以开源方式发布这些设计，以促进该领域的进一步研究：[2]

本文的组织结构如下：第2节回顾了面向边缘部署的最先进方案，展示了当前神经网络硬件主要关注8位精度、专用架构和稠密矩阵运算。第3节阐述了本研究的意义，并指出其技术特性与当前现有硬件的不同之处。第4节介绍了所提出的硬件架构，该架构支持可变精度浮点数/整数运算，可高性能执行稠密与稀疏张量运算。第5节评估了该架构映射到目标边缘设备 Zynq UltraScale+MPSoC 后的性能，并将其与脉动阵列设计、针对低精度张量运算优化的 ARM 库 RUY，以及参考 C++ 实现进行了比较。第6节介绍了动态重配置方法及其与软件定义设计框架和 TensorFlow Lite 推理引擎的集成。第7节通过不同的加速器

• 电气工程系的 *J. Nunez-Yanez*，瑞典林雪平大学，林雪平。电子邮箱：*jose.nunez-yanez@liu.se*
• *Andres Otero*、*Eduardo de la Torre*，工业电子中心，马德里理工大学，马德里，西班牙。电子邮箱：*joseandres.otero,eduardo.delatorre@upm.es*
稿件收到日期：*2005*年*4*月*19*日；修订日期：*2015*年*8*月*26*日。1. https://github.com/google/ruy

2. 网址https://github.com/eejlny/gemm-spmm

配置，评估其在 TFLite 中集成后的性能、功耗和能耗，并与针对同类边缘设备的其他最先进硬件进行比较。最后，第8节总结了本文并提出了未来工作。

2 相关工作与动机

2.1 灵活模型神经网络加速器

本节讨论无需更改硬件即可支持多个模型的神经网络加速器。利用灵活硬件加速 TensorFlow Lite 模型执行的工作，主要集中于采用用于张量运算的脉动阵列架构，以获得非常高的吞吐量。例如，谷歌提供了一种低成本、低功耗版本的谷歌 TPU，称为 EdgeTPU [2],，可运行采用 8位精度的专用神经网络。EdgeTPU 中的脉动阵列大小为 64 x 64 个乘加单元，在 480兆赫下可实现 4 万亿次运算每秒，其规模远小于 TPU 云端配置。采用浮点精度的层将在外部 CPU 上运行，该 CPU 将作为 EdgeTPU 设备的主机。Xilinx 也专注于推理，并通过赛灵思 DPU [3] 单元提供对 TensorFlow 的支持。该单元由寄存器配置单元、数据控制器和针对现场可编程门阵列硬件资源进行优化的卷积计算模块组成。原始硬件专用于卷积神经网络，但也提供了适用于其他模型类型的替代架构和硬件配置。DPU 硬件支持 4位、8位和 16位整数精度。Xilinx DPU 架构通常针对特定模型类型进行专门设计，并且需要供应商专用编译器、优化器和量化器，才能将 TensorFlow 模型部署到受支持的设备上。本文的方法更接近 EdgeTPU，而不是 Xilinx DPU，因为它无需额外的编译或优化即可应用于任何 TFLite 模型。EdgeTPU 仅支持 8位精度，而 FADES 支持 1位、2位、4位和 8位整数精度，以及单精度和半精度浮点。本文重点关注 8位精度和单精度浮点精度，这些精度与当前版本的 TensorFlow Lite 推理引擎兼容，该引擎运行在主机处理器上。

2.2 特定模型的神经网络架构

本节回顾了部署新模型时需要修改的硬件架构。大量研究聚焦于创建针对特定网络模型的工具和架构，例如简单的全连接神经网络（FCNN） [4],卷积神经网络（CNN）[5]或长短期记忆网络（LSTM） [6]。例如，[7],介绍了一种专用于权重稀疏化的 LSTM 类型网络的现场可编程门阵列架构。该架构在设计阶段确定稀疏矩阵向量运算中每一行所需的处理单元（PE）数量。研究还针对 LSTM 类型网络涉及的不同权重矩阵提出了一种双重剪枝策略。 [8] 还通过将随机计算原理融入 LSTM 模型，对 LSTM 类型网络进行了研究。该硬件采用随机数表示，并使用乘法运算采用异或非门，并额外使用多路复用器或并行计数器。FPGA 演示表明，与采用传统二进制算术的参考硬件相比，其功耗显著更低，且不会影响整体准确率。FPGA 中架构专用化的示例还包括 FINN [9]和 hls4ml [10]——这两个成熟的框架支持任意精度硬件。hls4ml 已证明，在粒子对撞机等延迟要求极低的问题上能够实现极高的性能。它采用量化感知训练和基于 QK-eras 库的剪枝，并将生成的配置导出为 C++ Vivado HLS 描述，以便在 FPGA 上实现。 [10] 中提出的神经网络示例由 3 个全连接层和一个 Softmax 层组成，精度为 14 位，其中包含 6 个整数位。剪枝会将许多权重置零，并移除与这些权重相关的硬件资源。剪枝不会影响延迟，因为非零权重决定了最长路径。因此，网络深度保持不变。其他针对 FPGA 映射进行优化的任务专用神经网络框架还包括 LUTnet [11]和 LogicNets [12]。这些框架进一步发展了利用 FPGA 查找表实现权重与激活值之间的 2 输入异或非运算这一概念（该方法用于二值网络），以发挥多输入查找表的能力。权重被固化在查找表所实现的逻辑函数中，而二值激活值则使用数量可变的查找表输入。通过剪枝技术可减少查找表所需的输入数量，从而提升性能并降低逻辑复杂度。无论如何，专用化意味着架构和最终实现均针对特定任务。

[13]讨论了用于加速现场可编程门阵列中卷积神经网络的不同硬件架构，并指出 CPU+FPGA 方法可以受益于仅将现场可编程门阵列高效处理的部分映射到 FPGA，而将更不规则的计算交由 CPU 完成。本文选择的正是这种异构方法，它还能够支持大型模型，而不需要大型 FPGA 器件。异构设备结合 CPU、现场可编程门阵列和图形处理器等不同处理资源，可支持低功耗边缘智能应用，并实现更好的集成；例如，[14] 提出的低延迟混合数据初始化改进了不同设备加载输入数据和网络参数数据的方式。专注于稀疏计算的加速器包括 [15] ，它与我们的工作有相似之处：稀疏权重采用压缩稀疏行格式存储，使用按形状剪枝，并采用同等级别的 UltraScale 器件，不过它使用 Verilog 而非高级综合进行开发。该加速器在 VGG16 上达到 990 GOPs/秒，DSP效率为 0.367 GOPS/DSP，并根据卷积神经网络模型使用不同的架构。异构设备也被用于稠密矩阵处理； [16] 提出了一种工具流程，用于对权重矩阵进行 8 位和 4 位值的动态精度量化，并根据层的精度为每层配置数量可变的处理单元。该设计采用经典的行缓冲区，针对 VGG 网络进行了优化，使用固定大小、包含 3×3 个值的窗口。我们的方法并非针对特定滤波器尺寸进行专门优化，而是旨在处理 TFLite 模型格式所用的任意矩阵形状。

3 研究动机

与第 2.2 节介绍的研究不同，本文的工作避免为特定神经网络模型构建专用架构，而是旨在直接以硬件实现替代 Google RUY 等软件矩阵乘法库，其概念与 Google EdgeTPU 类似。其优势在于，EdgeTPU 使用脉动架构，且仅支持 8位精度的稠密矩阵乘法；FADES 则能够高效处理多种精度以及稠密/稀疏模型，通过将数据流式传入内部先进先出队列和块 RAM 来尽量减少数据停顿，并且仅对块 RAM 内部存储器执行不规则数据访问。FADES 由一组动态可重构加速器覆盖层构成，可处理采用 TensorFlow 模型优化扩展工具包 [17]中的剪枝和量化技术进行优化的模型。这样，我们便能受益于 TensorFlow 社区在复杂量化感知训练和剪枝方面积累的大量研究成果。因此，本文的目标并非提出新的训练技术，以弥补稀疏度可能导致的准确率下降。在我们先前的工作 [18],中，我们研究了任意量化对长短期记忆网络和卷积神经网络层准确率的影响，并介绍了适用于小型 Xilinx Zynq 片上系统设备的独立稀疏与稠密计算硬件。该稀疏架构是矩阵向量硬件的扩展，因而限制了并行性，且未针对 TensorFlow Lite 提供专门支持。本文面向更大型的 Zynq Ultrascale 器件，提出了一种新型融合架构，可复用硬件资源，并集成到 TFLite 的推理引擎中。我们通过高级综合实现动态重配置，以支持浮点和整数精度。我们将该加速器的性能和功耗与虚拟重配置方案以及最先进的软硬件加速库进行比较。表1对比了若干选定的技术特性与 FADES 方案，以凸显两者的差异。

加速器	模型集在运行时	目标技术	动态浮点数-整数精度	稀疏支持灵活性		TensorFlow集成
EdgeTPU [2]	Yes	ASIC	No	No	Low	Yes
数据处理器 [3]	Yes	FPGA	No	Yes	Low	Yes
FINN [9]	No	FPGA	No	No	High	No
HLS4ML [10]	No	FPGA	No	No	High	No
LUTNET [11]	No	FPGA	No	No	High	No
[16]	No	FPGA	No	No	High	No
[15]	No	FPGA	No	Yes	High	No
FADES	Yes	FPGA	Yes	Yes	High	Yes

表1：硬件比较

4 融合架构

4.1 系统架构

低精度数据类型和稀疏神经网络表示是两种有利于在深度学习中采用现场可编程门阵列的研究趋势。受这些趋势启发，我们设计了 FADES 数据流架构，以加速 TFLite 矩阵运算，并支持非对称量化激活值、按列优先写入矩阵、逐滤波器/逐轴偏置值，以及当前 TensorFlow Lite 规范中的缩放功能。FADES 计算深度神经网络模型的张量运算，在所测试的神经网络中，这部分运算最多占执行时间的 95%。在 FADES 中，我们采用分块方法设计了 SPMM（稀疏矩阵乘法）和 GEMM（通用矩阵乘法）加速器，以处理由大量列组成的 B 矩阵。我们使用高级综合（HLS）描述来定义架构。要同时支持 GEMM 和 SPMM，面临的挑战是在 HLS 描述中避免不必要的硬件复制和性能下降。图 1 展示了架构端口的顶层框图。模式输入将硬件设为稀疏或稠密模式；数据端口传输输入矩阵 A、B 和输出矩阵 C 的数据值；配置端口则控制 TensorFlow Lite int8 缩放和截断参数、可选偏置及矩阵维度。正在处理的矩阵形状由运行时参数定义： $N = A$ 行、 $M = A$ 列 和$P = B$ 列。

图2展示了由4个阶段组成的数据流内部组织，这些阶段通过流式先进先出缓冲区互连，适用于通用矩阵乘法和稀疏矩阵乘法。所有数据流阶段均使用高级综合编写，以实现高效的初始化间隔（例如，II为1）。在高级综合中，初始化间隔为1意味着每个时钟周期都可以启动循环的新一次迭代，这是实现高性能的关键要求。在FADES中，流式架构意味着先将稠密矩阵的一个分块缓冲在FPGA器件内部，然后将压缩稀疏行格式的稀疏矩阵从存储器流式传输至加速器。稀疏值在流式传输过程中，会根据分块大小与稠密矩阵的多个值进行计算。处理完一个分块后，系统会载入新的分块，并再次传输稀疏数据。这种方法使所有不规则访问都发生在本地块随机存取存储器中，不会造成停顿或额外延迟。

第1阶段负责加载一个 B 矩阵块，其列数等于 $PEs*4$ 字节（每个处理单元处理四个 8 位元素或一个单精度浮点元素），行数等于 B 矩阵中的行数。随后，它开始以原始格式为通用矩阵乘法传输 A矩阵中的元素；对于稀疏矩阵乘法，则以压缩稀疏行格式传输元素，并附带列索引 和非零 值。这意味着在稀疏矩阵乘法模式下，加速器会针对每个矩阵读取数量不定的权重值和列索引。在通用矩阵乘法模式下，读取次数是固定的，由行大小乘以权重矩阵的行数确定；而在稀疏矩阵乘法模式下，读取次数则由权重矩阵中的非零元素数量确定。这不会成为数据流架构中的性能瓶颈，因为读取阶段独立于其他阶段，只要不存在缓冲区下溢或溢出，它就能以全速处理 A 矩阵中的所有数据元素，无论是通用矩阵乘法还是稀疏矩阵乘法。

阶段 2 是主要的计算循环，会实例化大部分 DSP 块。其目标是在每个时钟周期内并行激活所有处理单元。对于所有列数大于或等于 $B\ _WIDTH\ _BLOCK$ 指定的处理单元数量的 B 分块，情况皆是如此。 $B\ _BLOCK\ _WIDTH$ 是与并行性相关的主要参数，它定义了

并行处理激活矩阵B 各列的处理单元数量。通常，最后一个分块包含的列数少于 $B\ WIDTH\ BLOCK$，在这种情况下，部分处理单元不会写入其输出先进先出队列。这使得系统能够支持形状任意且列数不是分块大小整数倍的矩阵。

阶段3使用 QM（量化乘数）和 $shift$，按照 Tensorflow Lite 对 int8 值的规范实现缩放策略。它读取阶段 2 生成的原始值，并将相应的缩放值写入第4阶段。浮点模式下，阶段3会将原始值直接转发到第4阶段。最后，第4阶段读取这些值，并将它们写入正确的内存地址，从而将多个分块的输出组合起来，正确构造输出矩阵 C。

该加速器提供多种编译时配置选项，汇总于表2中。表中的各列分别表示核心数（CC）、每个核心的处理单元数（PE）、每个核心的并行行数（PR，用于设定并行处理的 A 矩阵行数）、支持的模式（仅 SPMM、仅通用矩阵乘法或同时支持 SPMM 和通用矩阵乘法的 FUSED）、是否启用输出矩阵转置（TRANS），以及是否启用缩放（SCALE）。SCALE 可确保结果符合 TFLite 规范，使用逐滤波器量化乘数参数。逐滤波器量化乘数参数是近期 TFLite 版本的一部分，可为每个滤波器启用不同的缩放参数，从而提高准确率。TRANS 可确保结果矩阵以列主序格式写入，因为 TensorFlow Lite 推理引擎目前要求采用这种格式。CC 选项会在设备中实例化多个相互独立的核心，并在这些核心间均匀分配计算任务。分配方式可以是采用多个 A 矩阵块和一个 B 矩阵，也可以是采用一个 A 矩阵和多个 B 矩阵块。第一种方式会将稀疏矩阵 A 划分为多个块；本文采用这种方式，以便通过多个端口优化对 A 值和列索引的读取。

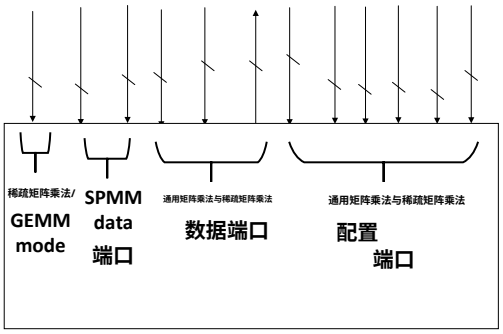


图 1：顶层框图

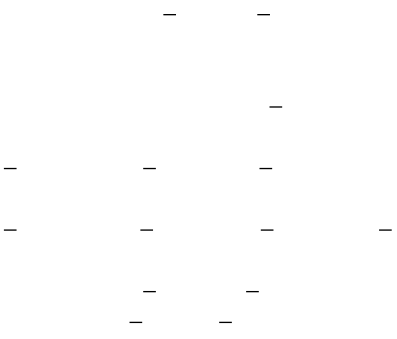
CCs	PEs	PRs	SPMM	GEMM	转换	缩放
1 到 4	2 到 256	1 到 2	0 到 1	0 到 1	0 到 1	0 到 1

表2：加速器配置选项

4.2 浮点与 int8 硬件支持

在 FADES 设计中，每个数据字宽为 32 位，由权重值/激活值构成，其位宽取决于所需精度。在我们之前的工作中，我们展示了如何将 32 位值解释为多个 2、4 或 8 位整数值 [18]。本文重点分析与运行在 PS 上的 TFLite 推理引擎直接兼容的 *int8* 和浮点模式。尽管通常认为 8位精度足以满足深度学习推理的需求，但网络中的某些层或某些类型的网络（如循环神经网络）可能需要浮点精度 [19]。当应用需要通过网络再训练而不仅仅是推理来适应输入数据统计特性的变化时，浮点精度也很有用。列表1和列表2分别展示了 ***float*** 和 *int8* 精度下计算函数的代码片段。请注意，$int8$ 列表展示了一个处理单元中得到的四个结果的部分累加，而浮点列表没有进行这种累加，因为每个处理单元只有一个结果。在这两种情况下，未展示的外层循环都会对激活矩阵 B 的所有行上的结果进行累加。

列表1：浮点计算内核示例 1 2 f o r (i n t j = 0; j < B WIDTH BLOCK ; j ++)3 {4 # pragma HLS UNROLL 5 # pragma HLS PIPELINE 6 ap_int <32> b b l o c k i n t =b block [↪ b row] [j]; 7 ap uint <32> A val =a value ; 8 FTYPE a v a l f l o a t =* (FTYPE *)&A val ; 9 FTYPE b v a l_f l o a t =* (FTYPE *_)&↪ b b l_o c k i n t; 10 f l o a t r h s f l o_a t =f l o a t (↪ z e r o p o i n t r h s) ; 11 acc [j_] =(a v a l f l_o a t) * (_b v a l f l o a t −↪ r h s f l o a t) ;12 }// j 循环 清单2：int8 主计算内核 1 2 f o r (i n t j = 0; j _< B WIDTH BLOCK ; j ++) {3 # pragma HLS UNROLL_4 # pragma HLS PIPELINE 5 f o r (i n t z = 0; z < DTYPE LENGTH ; z↪ +=8) {6 ap int <8> A val =a value_. rang e (z↪ +7 , z) ; 7_ap int <8> B val =b block [b row] [↪ j]. range (z +7 , z) ; 8 acc [j] += A val * (B val −↪ z e r o p o i n t r h s) ; 9 }10 }// j 循环



为了最大限度地复用硬件，编写代码时尽量减少对数据精度有依赖的差异。接口和缓冲存储器中的数据类型主要定义为 *uint32* ，只有在计算内核中才使用指针

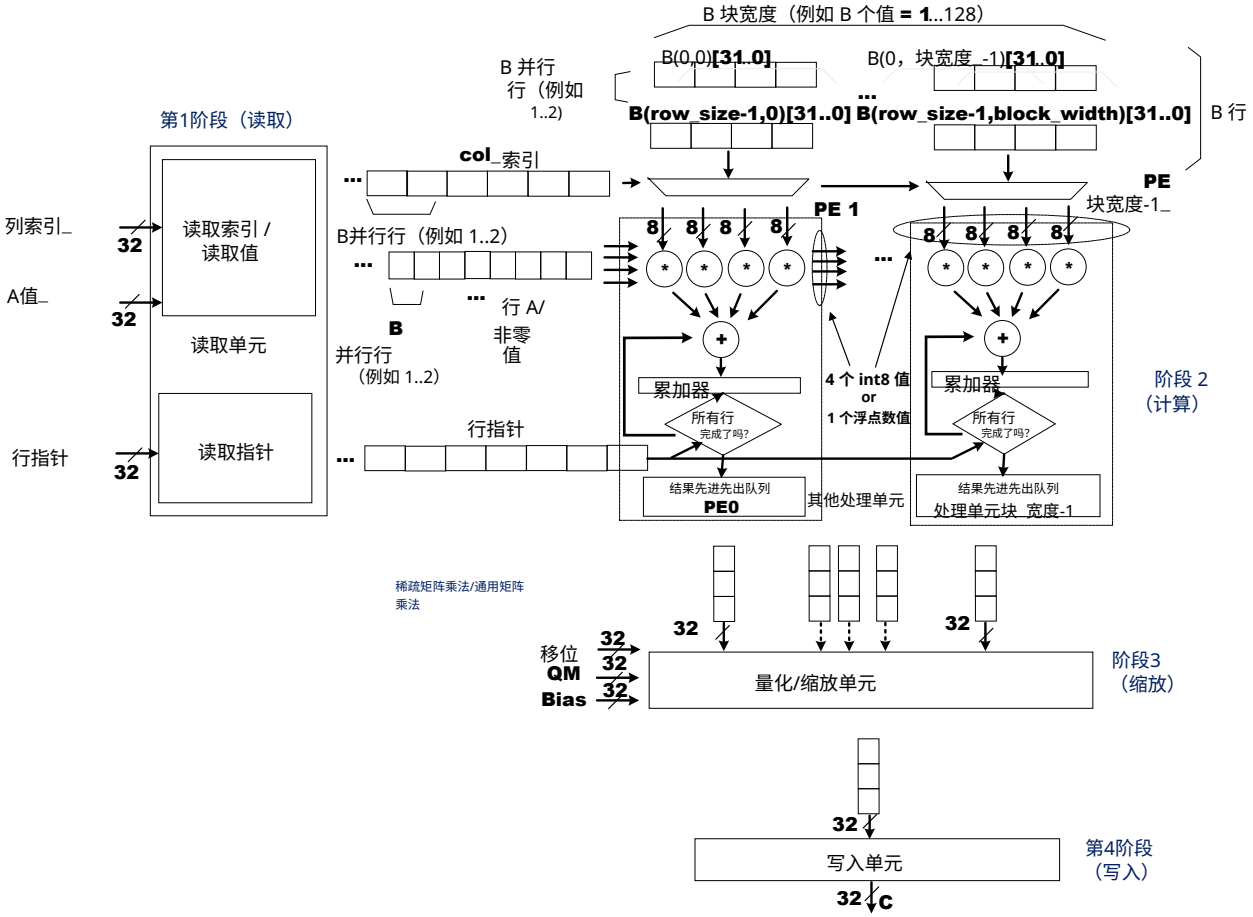


图 2：融合通用矩阵乘法/稀疏矩阵乘法架构组件

重新解释，将 *uint32* 值解释为四个*int8* 分量，或解释为单个浮点数库值，概念上如列表1第7行和第8行所示，其中*A val* 是一个 *uint32* 值，而*a val float* 是一个浮点数值。对*a val float* 执行的运算将是标准浮点运算。

要在浮点模式下获得高性能，仅将 32 位字解释为浮点值是不够的。问题在于，Zynq Ultrascale 中用于累加的浮点加法（FADD）延迟大于一个时钟周期。因此，要实现每个时钟周期完成一次加法，必须用交错操作填满 FADD 内部的流水线。另一方面，整数加法的延迟为一个时钟周期，因此 C综合工具无需交错操作即可实现为 1 的初始化间隔。为了使浮点内核也达到等效的 1 启动间隔，我们在同一处理核心上交错执行 FADD LATENCY 次部分累加，每次累加每经过 FADD LATENCY 个时钟周期完成一次。C综合工具能够识别出，可以在交替的时钟周期将这些部分累加调度到同一个加法器核心上。FADD LATENCY 的最优值会随目标频率提高而增大；在 200 MHz 时，其值为 6。200 MHz 是所有运行于 Zynq UltraScale+设备上的实现所选用的频率。由此得到的架构每 32 位使用 4 个 DSP48模块

每个浮点运算符和每个*int8* 运算符分别使用 1 个 DSP48 块，因此 32 位浮点配置和 32 位 int8配置所使用的 DSP48模块数量相近。表3 展示了不同配置的复杂度，具体如下。每对数字中的第一个数字表示核心数量，在本例中始终为 1；第二个数字表示处理单元数量，可以是 32 或 128。需要注意的是，浮点版本和*int8* 版本中的 DSP 块数量有所差异，这是因为*int8* 配置中包含缩放模块，而浮点配置中增加了交错的部分结果。表中的第三行展示了同时支持 int8 和浮点配置的设计复杂度：使用多路复用器来选择启用浮点数据通路还是 *int8* 数据通路。源代码中的指针重新解释使两种配置的代码保持相似，并最大限度地实现逻辑复用。无论如何，1 次浮点运算和 4 次 int8 运算所对应的 DSP48模块内部组织不同，因此同时启用两种配置时，编译器需要复制 DSP48 的使用。这种额外的资源使用量意味着，例如，128 浮点-int8配置无法满足 200 兆赫下的时序要求，尽管单独的配置均能毫无问题地满足时序要求。动态功能交换的主要优势在于，无需同时部署两种配置。第6节将探讨动态功能交换如何

降低资源使用量并改善功耗（因为只需配置参与计算的逻辑单元），同时提升性能（因为可以创建包含更多处理单元的处理核心，而无需降低时钟频率）。

配置	查找表(K)	触发器(K)	18K 块 RAM	DSP48E
(1,32) 浮点数	90.9	120.5	186	181
(1,32) int8	50.8	56.9	189	161
(1,32) 浮点-int8	96.8	121.6	187.5	325
(1,128) 浮点数	198.2	245.9	552	661
(1,128) int8	67.2	78.7	571	545
(1,128) 浮点-int8	221.8	260.8	564	1189
(32x32) 脉动阵列 int8	68.2K	27K	1041	1031

表3：配置复杂度比较

总而言之，整个架构基于不同功能或阶段构成的数据流。随后，每个阶段采用流水线处理，这可以视为细粒度数据流（或者等价地，数据流可以视为粗粒度流水线）。当运算对象为浮点值时，浮点加法的延迟超过一个周期，因此浮点加法累加的流水线处理会遇到问题。为此，本节所述的交错技术被用于使浮点加法的整体延迟达到一个周期。其副作用是浮点逻辑与整数逻辑存在显著差异，因此将采用动态重配置在不同模式之间切换。

5 初始性能与功能验证

5.1 实验设置

本节将验证 FADES 处理核心相较于高性能脉动式硬件 [20] 的性能特征。此次初步性能评估针对独立实现进行，尚未将其集成为 TFLite 框架的一部分，以便与 [20]进行公平比较；第 7 节将分析 TFlite 集成后的性能。 [20] 和所提出的设计均使用 Xilinx SDSoC 2018.3 实现，可编程逻辑的时钟频率为 200 兆赫，处理系统则以标准频率 600 兆赫运行。为验证设计功能正确，我们创建了一个使用 RUY 乘法库编译的 C 测试平台。矩阵以随机数据初始化，并将硬件生成的输出与 RUY 库产生的软件结果进行比较，以确保正确性。脉动架构在同一块配备 Zynq UltraScale 器件的 ZCU102 开发板上实现，并运行 Ubuntu 16.04。脉动核心包含 32x32 = 1024个处理单元，每个处理单元按照原始设计使用一个 DSP 块。我们还将高性能乘法库 RUY 3 作为对比基准。该库由谷歌工程师开发，旨在满足神经网络推理引擎的矩阵乘法需求。RUY 可作为一个选项用于 TFLite 的 ARM CPU 架构上。它的设计目标不仅是在超大矩阵上实现高性能，也要针对 TensorFlow Lite 应用相关的矩阵尺寸和形状实现高性能。RUY 完全使用汇编代码来利用 ARM NEON 和 SIMD 优化，并同时支持浮点矩阵和 8 位整数定量化矩阵。它取代了 EIGEN 和 GEMMLOWP，性能更佳。RUY 的替代方案之一是 XNNPACK，但 XNNPACK 仅专注于浮点运算，因此不适用于本研究。XNNPACK 还依赖于 armv8.2 指令集，而 Zynq UltraScale+ MPSoC 中使用的 Cortex-A53 等 armv8 处理器并不支持该指令集。脉动式硬件仅支持 int8，而 RUY 可配置为使用浮点数和 int8 精度。我们首先考虑采用稠密模式处理方阵，因为脉动架构和 RUY 均无法在稀疏模式下运行。

5.2 初始性能结果

表4展示了这些结果，其中包括一个参考 C++ 实现；每个测试点均已验证输出正确。表中显示，在 int8 模式下，采用 (1,128) 配置的 FADES 明显优于脉动式硬件和 RUY，尤其是在处理的矩阵规模增大时。这一点意义重大，因为脉动阵列配置包含 32x32 个单元，DSP 模块数量是其两倍。显然，C++ 实现的速度远慢于经过软件优化的 RUY。总体而言，对于 1024x1024 的矩阵，FADES (128,1)/int8 在 int8 模式下比 RUY 快 18 倍，而 FADES (128,1)/float 在浮点模式下快 9 倍。请注意，FADES 加速器包含 TFLite 所需的逐滤波器缩放硬件，因此，与仅执行矩阵乘法的其他通用矩阵乘法硬件直接比较时，必须考虑这一点。

	128	256	512	1024
脉动阵列 int8	0.29	1.27	5.34	21.92
RUY int8	1.53	6.14	41.03	302.31
C++ int8	547.4	4k	35k	314k
1,128 int8	0.307	1.001	3.618	16.58
1,32 int8	0.365	1.575	8.65	54.78
RUY 浮点数	1.94	14.73	64.63	500.02
C++ 浮点数	583.48	4651.59	38923.8	423516
1,128 浮点数	0.089	3.28	12.88	56.23
1,32 浮点数	0.921	4.761	29.33	200.5

表4：方阵的浮点数/int8 性能（毫秒）

(1,32) 配置使用的 DSP 模块数量减少了 4 倍；对于较大的矩阵，其速度约为 (1,128) 配置的四分之一，不过如预期所示，这种性能差异在较小的矩阵上会有所减弱。(1,32) 配置的资源使用量较低，因此在 DSP 使用量相同的情况下，可将更多核心映射到 Zynq Ultrascale 设备上。表5对一个大矩阵进行了性能比较，比较了 (1,128) 方案（一个包含 128 个处理单元的核心）和 (4,32) 方案（4 个核心，每个核心包含 32 个处理单元），并采用稀疏模式及 3 个稀疏度级别：低（稀疏度为 50%）、中（稀疏度为 70%）和高（稀疏度为 90%）。请注意，(1,128) 和 (4,32) 配置包含相同数量的处理单元（即 1x128 = 4x32 = 128 个处理单元）。显然，对于稠密矩阵和稀疏度较低的情况，多核配置并不能显著提升性能。原因在于，对于稠密矩阵，计算强度较高，而瓶颈在于计算能力；两种配置的处理单元数量相同，因此计算能力相当。另一方面，随着

3. https://github.com/google/ruy

稀疏度增加，性能瓶颈转向数据移动，而多核配置提供的额外端口则带来了明显更好的性能。与重新配置整个设备相比，(1,32) 配置中采用较小的核心，也更有利于引入 DFX 策略。后续章节将探讨采用 (1,32) 核心的 DFX 方法在性能和功耗方面的优势。

	稠密（毫秒）	稀疏 低	稀疏 中	稀疏高
1,128,[1 int8]	32.776	19.75	13.51	12.44
4,32,[1 int8]	28.54	18.68	9.3	4.95

表5：单核与多核性能（ms）

6 DFX 方法

DFX 方法基于 Xilinx [21] 最初提出的工作，用于实现基于 *Pblock* 和黑盒的动态可重构系统。我们将其扩展，以支持包含加密 IP 核（如浮点运算器）的软件定义系统。*Pblock* 定义了与可重构分区关联的规划区域，而每个可重构分区又包含多个可重构模块或变体。在该系统中，可重构分区有两个变体，分别对应 *int8* 和 浮点。黑盒用于在设计的静态部分指定可重构分区。在本工作中，我们不使用 Vivado 中提供的 DFX 向导，因为它通过在框图中选择整个 IP 核作为可重构分区来实现配置。在我们的案例中，我们采用更细的粒度，因此在高级综合工具生成的 RTL 中指定可重构区域。其思路是仅选择需要重新配置以便在 *int8* 和浮点之间切换的功能单元所对应的 RTL。

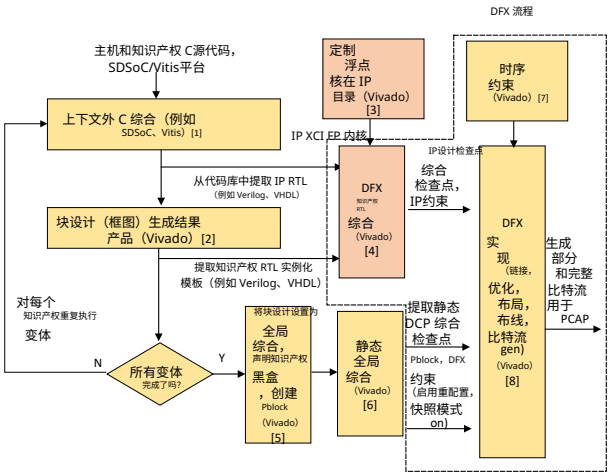


图 3：高级综合 DFX 方法

图 3 展示了所提出并采用的替代方法，其中的步骤编号如下所述：

- 步骤 1：高级综合输入代码在 Xilinx 的 OOC（独立于上下文）模式下进行初始 C综合，在此模式下，每个 IP 核都独立进行综合。随后，可在相应的代码库中获取该 IP 的 RTL 代码。
- 步骤 2：生成输出产品后，可在 Vivado 中获取该知识产权核的 RTL实例化模板。该实例化模板和代码库中的知识产权核 RTL 是定义 DFX 流程中知识产权变化形式的输入（即 *int8* 或浮点数）。
- 步骤 3：如果该 RTL 知识产权核实例化了浮点核心，则需要使用 Vivado IP集成器生成这些核心，以避免在后续实现阶段出现黑盒错误。这是因为浮点库已加密，不能直接用作附加 RTL 文件。
- 步骤 4：标准 DFX 流程会读取 RTL，并对该知识产权核的每个变体执行 RTL 综合，为每个变体生成一个 DCP检查点，供 DFX 实现流程的其余组件使用。

第 3 步中的 [5] 和 [6] 对应于在布局规划器中创建 Pblock、为设计的动态部分声明黑盒以及执行静态综合。最后，在 [7], 中定义时序约束，标准 DFX 实现流程 [8]会对每个变体进行布局、布线并生成比特流文件：除了将设计的静态部分和动态部分链接后生成完整比特流外，还会生成部分比特流。

扩展名为 .bit.bin 的比特流文件可由 Zynq 设备中的处理器配置端口（PCAP）使用，使处理系统能够加载完整或部分文件。除了这种硬件流程外，软件流程还会生成一个硬件库，以便从处理系统调用硬件执行。各硬件变体的 API 完全相同，因此可以将单个硬件接口库与软件应用程序的其余部分链接。为了在 TensorFlow Lite 中集成 DFX，我们利用 TensorFlow 2.7 支持的选择性层量化功能。推理期间，主机会检测该层是否有可用的缩放参数，以判断其是否为量化层；随后，在必要时检查并配置可编程逻辑，使其载入正确的知识产权（IP）。必须将正在重配置的区域保持在隔离状态，以避免部分重配置期间出现不可预测的活动并影响静态部分。在本研究中，我们通过 Linux内存映射函数 MMAP，将处理系统（PS）中相应的时钟复位物理寄存器映射到用户空间，从而使处理核心区域及其接口保持复位状态。对于高级综合设计中数量众多的接口信号而言，与使用部分重配置（PR）解耦器知识产权（IP）为 PR 设计提供逻辑隔离能力相比，这是一种更简单的替代方案。

7 FADES 在 TFLite 中的验证

本节将集成 FADES 到 TFLite 框架后，以性能、功耗和能耗特性作为验证参数进行分析。该处理核心可直接替代对 RUY库的调用。TFLite

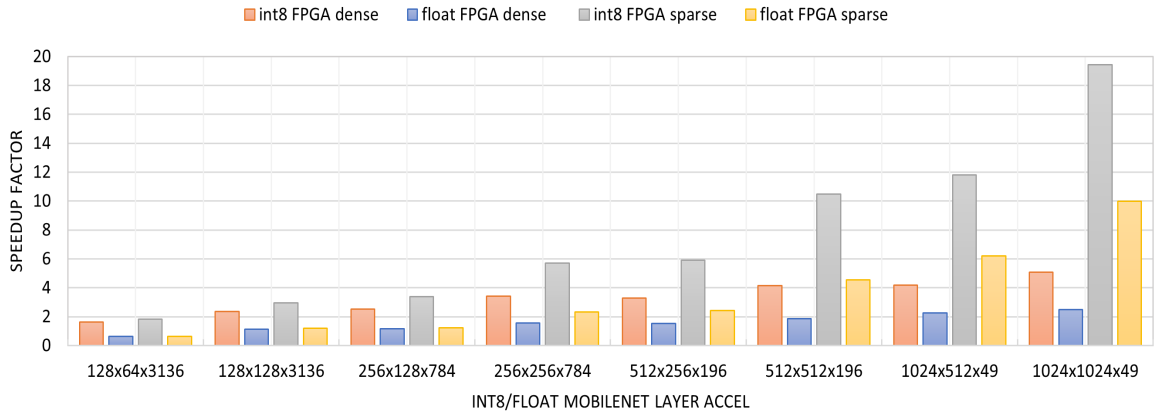


图 4：TensorFlow Lite 中 1、32 的性能分析

所用版本为 2.7，并已从源代码编译，运行于 Zynq Ultrascale 器件上。除了 DFX（动态功能交换）之外，我们还将虚拟功能交换（VFX）和完全重配置（FR）纳入比较。在 VFX 中，硬件同时实现 浮点数 和 *int8*模式，并使用多路复用器选择启用的路径。这种虚拟模式意味着硬件需求会增加，但模式切换只需一个时钟周期。表 6 对比了 VFX 和 DFX 模式的硬件复杂度。

配置	查找表（K）	触发器（K）	18K 块 RAM	DSP48E
浮点 DFX	90.9	120.5	186	181
int8 DFX	50.8	56.9	189	161
浮点-int8 VFX	96.8	121.6	187.5	325

表 6：DFX/VFX 硬件复杂度比较

表 6 显示，VFX 所需的额外查找表/触发器数量不多，但 DSP 数量几乎翻倍。额外的硬件需求意味着，部署 VFX 会限制可用并行核心的数量，并且与 DFX 相比还会增加功耗/能耗。FR 模式对应于一次完整的比特流重配置，用浮点硬件替换*int8*硬件。所考虑设备的比特流大小为 25MB，我们测得执行此次完全重配置所需的时间为 200ms。另一方面，部分核心的比特流大小为 9MB，部分重配置时间约为 30ms。因此，采用完全重配置所产生的能耗和时间开销将接近部分重配置的 7 倍。除了在时间方面 DFX 优于 FR 之外，未进行重配置的 DFX 核心还可以继续运行，因此在完全重配置期间，系统无需完全停止运行。

7.1 FADES 性能分析

图4展示了单个32处理单元（1,32）配置的性能，并将其与RUY在从MobileNet神经网络中提取的层上所提供的性能进行了比较。显然，对于*int8* 和浮点模式，较大、较宽的层都能从加速中获益更多，但该硬件可以加速所有MobileNet层。如第5节针对大型层所示，采用多个现场可编程门阵列核心的配置可显著缩短执行时间。*int8* 配置获得的加速比高于浮点硬件，这可以归因于浮点模式下RUY软件的良好性能：其速度并非比RUY*int8*慢4倍。RUY使用汇编代码，利用NEON SIMD加速器执行整数和浮点运算。另一方面，现场可编程门阵列浮点硬件通常比*int8*硬件慢4倍，因为每个浮点值包含四个*int8*值。稀疏配置采用了90%的高稀疏度，其加速效果显著优于稠密配置。训练过程中使用TensorFlow模型优化工具包当前具备的功能进行稀疏化，并且每次只对一层进行稀疏化，以尽量减小对准确率的影响。我们在TensorFlow中通过模型克隆和选择性剪枝函数来选择剪枝层。我们采用多项式衰减策略，逐步将剪枝比例从50%提高到不同的最大值。本文旨在展示硬件加速稀疏层的能力；稀疏化对准确率的影响不在讨论范围内。*int8* 硬件所实现的加速比高于浮点硬件。需要注意的是，得益于DFX实现的硅复用能力，无需同时配置浮点模式和*int8* 模式。这意味着最多可采用4核配置，性能优于不可重配置的替代方案（VFX）；在后者中，为了容纳同时支持浮点模式和*int8* 模式的硬件，必须减少核心数量。

在表7中，我们估算了 FADES 的性能和硬件细节，并与第 2 节中综述的、可用于稀疏和稠密处理的其他嵌入式硬件进行了比较。我们给出了单处理核心（x1）和四处理核心（x4）配置的结果，并根据当前模式为浮点数还是 int8，分别报告了两种复杂度和性能值。表中显示，DSP效率与文献中的最佳数值相当，而原始 GOPS 与该配置使用的 DSP 模块数量成正比。FADES 相较于

	[15]	[15]	[16]	[16]	[20]	我们的 x1	我们的 x4
设备	XCZU9EG	XCZU9EG	XC7Z045	Nvidia TK1	XCZU9EG	XCZU9EG	XCZU9EG
Type	稀疏	稀疏	稠密	稠密	稠密	稀疏/稠密	稀疏/稠密
频率	200	200	150	852	200	200	200
精度	16 位	8bit	16 位	浮点数	8bit	8 位/浮点数	8位/浮点数
DSPs	1350	2520	780		1031	161/181	644/724
LUTs	390k	405k	182k		142k	50.8K/90.9K	203k/363k
块 RAM	1460	1460	486		528	189/186	700/688
DSP效率（Gops/DSP）	0.36	0.39	0.23		0.18	0.47/0.11	0.47/0.11
性能（Gops）	495	990	187	76	195	76/20	303/75

表7：与其他嵌入式硬件的比较

这些硬件的主要区别和优势在于，它结合了稠密-稀疏模式，并能通过动态重配置在 int8 和浮点精度之间切换。另一项区别是，它集成于 TFLite，并支持 TFLite 规范中的缩放模式和非对称激活。

7.2 FADES 功耗与能耗分析

ZCU102 开发板的现场可编程门阵列中，每条电源轨都配有分流电阻。这些分流电阻连接到德州仪器 INA226 器件，可用于测量电压和电流。这些器件连接到 I2C 总线，可通过 Xilinx Petalinux 内核中的 */sys/class/hwmon*接口访问。我们创建了一个应用程序，用于读取与 PS 和 PL 对应的功耗值，并生成功耗值轨迹。由于功耗测量装置的限制，每 6 ms 可获取一个功耗采样。

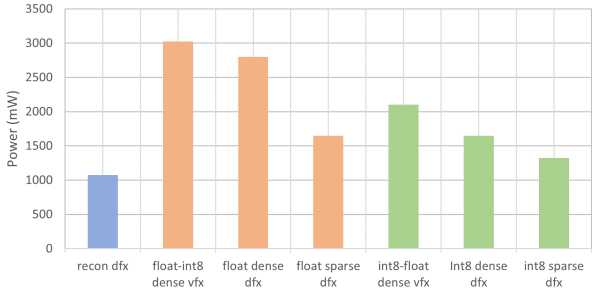


图 5：（1,32）不同配置下的功耗分析

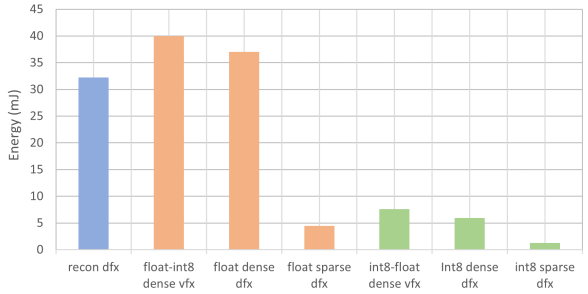


图 6：（1,32）不同配置下的能耗分析

图 5 展示了 PL 中的功耗评估结果，表明在重配置期间，功耗较低，因为在这些运行过程中，没有进行任何计算，PS 仅加载配置存储器。在所有这些运行过程中，PS 的功耗基本保持恒定，约为 2 瓦。功耗最高的模式是浮点 -int8 和 int8- 浮点模式，它们分别对应于配置为以浮点模式或 int8 模式进行计算的 VFX 硬件。在 DFX 中，浮点模式硬件的功耗高于 int8 硬件，这主要是因为流水线使用了额外逻辑来隐藏浮点运算较高的延迟，如表3所示。请注意，浮点加速器使用的额外逻辑使 int8 和浮点操作数的延迟均达到一个时钟周期，如第 4.2 节所示。与稠密配置相比，稀疏配置的功耗也明显降低。随着输入A 矩阵稀疏度增加，瞬时功耗需求随之下降。这可归因于 PL 中的稀疏计算强度较低，即每从存储器读取一个字节所需的运算数较少，因此 DSP 利用率也较低。

为验证这一 DSP 利用率，我们估算了 FADES 加速器的理论最大吞吐量，假设核心数据传输和 DSP 逻辑均达到 100% 饱和。在稠密模式运行中，我们测得数据移动和算术流水线的饱和度均超过 90%。随着稀疏度增加，这一利用率会下降；数据移动成为瓶颈，导致 DSP 模块的利用率降低。这是因为，尽管稀疏度降低了稀疏矩阵 A 所需传输的数据量，但稠密矩阵 B 仍需完整传输，即使输入的稀疏度为 100% 也是如此。这种较低的硬件利用率可从图 5 中观察到的功耗下降看出。

最后，图6比较了 Mobilenet 层（1024x1024x49）在不同配置下的能耗需求，以及动态重配置能耗。图中显示，能耗最高的配置是使用 VFX 进行密集计算。与 VFX 相比（处理单元数量相同），DFX 稠密配置能够保持执行时间不变，但由于逻辑规模缩减，能耗更低。稀疏DFX配置得益于执行时间和功耗的降低，成为能效最高的运行方式。与浮点数配置相比，int8 配置也受益于更低的功耗和更短的执行时间。图中还显示，浮点硬件与 int8 硬件之间的动态重配置过程所需能耗

并非微不足道，但对于包含数十层的大型网络而言，只要重配置速率不是很高，这种开销仍可接受。例如，浮点硬件仅用于训练，或随时间用于若干组层时。

8 结论与未来工作

本研究得出以下结论：

- 将稀疏模式和稠密模式整合到同一架构中，可以在层级选择执行模式，同时将开销降至最低。如果在训练期间确认稀疏模式不会对准确率产生负面影响，就可以将相应层配置为以稀疏模式运行。即使稀疏度仅约为 50%，稀疏模式仍比稠密模式更快，这得益于其数据流架构能够并行读取 CSR 值和索引。

- 可以采用动态功能交换，针对不同的 TFLite*int8* 和浮点精度优化所使用的 DSP 模块。这种硬件优化通过将更多核心映射到单个设备来提升性能，同时还能降低功耗，避免同时配置两条流水线。

- 对于浮点数和 *int8* 精度，稀疏模式的性能都优于稠密模式；此外，稠密硬件也显著快于软件优化的稠密 RUY 库，以及采用相同技术的脉动阵列实现。

- 动态重配置时间测得为 30 毫秒，这意味着按层进行重配置的速度太慢。更合适的做法是在推理与训练模式之间切换，或切换网络模型时进行动态重配置。

- 加速器需要缓冲一个稠密矩阵分块，其宽度取决于处理单元的数量，深度则等于行数。对于非常大的矩阵，块 RAM 可能无法容纳所有行，此时需要在软件层面进行分块。

未来工作可以探讨以下几个研究方向：

- 本研究重点关注浮点和 *int8* 精度，因为 TensorFlow Lite 原生支持这两种精度。目前正在引入对 16 位浮点数的支持，其他研究（如 hls4ml [10] ）则探索了二进制、三进制和四进制等亚字节精度，尽管这些精度目前尚未纳入 TFLite 推理引擎。随着 TFLite 支持的算术精度种类增加，DFX 有望更有助于避免资源开销。

- 我们已验证 DFX 在单核下的功能及优势。进一步探索多核配置将很有意义：在这种配置中，未进行重配置的核心仍保持活动状态，并执行有用的计算。

- 在当前版本中，加速器的配置由设计人员手动完成。我们需要一个调度器，作为推理引擎的一部分运行在主机处理器上，并向加速器发出重配置命令。该调度器需要根据准确率和性能测量结果，使用预测模型为每一层决定最佳精度和稀疏度。

- 最后，探索 FADES 加速器在不同精度下用于运行时训练，以及应用于变换器和循环网络等其他网络类型，是未来的研究方向。

致谢

本研究部分获得英国皇家学会产业奖学金、INF\R2\192044 MINET、EPSRC HOP-WARE EP\RV040863\1、Leverhurme Trust 国际奖学金 IF-2021-003，以及 Knut and Alice Wallenberg Foundation 资助的 Wallenberg AI Autonomous Systems and Software（WASP）项目的支持。

参考文献

[1] R. David、J. Duke、A. Jain、V. J. Reddi、N. Jeffries、J. Li、N. Kreeger、I. Nappier、M. Natraj、S. Regev、R. Rhodes、T. Wang 和 P. Warden，“TensorFlow Lite Micro：面向 TinyML 系统的嵌入式机器学习”，2020 年。[2] M. G. S. Murshed、C. Murphy、D. Hou、N. Khan、G. Anantha-narayanan 和 F. Hussain，“网络边缘的机器学习：综述”，2020 年。[3] J. Zhu、L. Wang、H. Liu、S. Tian、Q. Deng 和 J. Li，“一种用于加速 FPGA 上基于数据处理器的卷积神经网络推理的高效任务分配框架”，*IEEE Access*，第 8 卷，第 83224–83237 页，2020 年。[4] L. F. S. Scabini 和 O. M. Bruno，“全连接神经网络的结构与性能：新兴的复杂网络特性”，2021 年。[5] G. Lentaris、G. Chatzitsompanis、V. Leon、K. Pekmestzi 和 D. Soudris，“结合算术近似技术以改进 CNN 电路设计”，2020 年。[6] S. A. Ghasemzadeh、E. Bank-Tavakoli、M. Kamal、A. Afzali-Kusha 和 M. Pedram，“BRDS：一种基于 FPGA、采用行平衡双比率稀疏化的长短期记忆网络加速器”，*CoRR*，第 abs/2101.02667 卷，2021 年。[7] S. A. Ghasemzadeh、E. B. Tavakoli、M. Kamal、A. Afzali-Kusha 和 M. Pedram，“BRDS：一种基于 FPGA、采用行平衡双比率稀疏化的长短期记忆网络加速器”，2021 年。[8] G. Maor、X. Zeng、Z. Wang 和 Y. Hu，“基于随机计算的长短期记忆网络的 FPGA 实现”，载于*2019 IEEE 第 37 届国际计算机设计会议（ICCD）*，第 38–46 页，2019 年。[9] Y. Umuroglu、N. J. Fraser、G. Gambardella、M. Blott、P. Leong、M. Jahre 和 K. Vissers，“Finn”，*2017 年 ACM/SIGDA 国际现场可编程门阵列研讨会论文集*，2017 年 2 月。[10] J. Duarte、S. Han、P. Harris、S. Jindariani、E. Kreinar、B. Kreis、J. Ngadiuba、M. Pierini、R. Rivera、N. Tran 等，“面向粒子物理学的 FPGA 深度神经网络快速推理”，*仪器学报*，第 13 卷，页码 P07027–P07027，2018 年 7 月。[11] E. Wang、J. J. Davis、P. K. Cheung 和 G. A. Constantinides，“LUTNET：学习 FPGA 配置以实现高效的神经网络推理”，*IEEE 计算机汇刊*，第 69 卷，第 1795–1808 页，2020 年 12 月。[12] Y. Umuroglu、Y. Akhauri、N. J. Fraser 和 M. Blott，“利用 LogicNets 实现高吞吐量深度神经网络推理”，载于*2020 IEEE 第 28 届年度国际现场可编程定制计算机研讨会（FCCM）*，（美国加利福尼亚州洛斯阿拉米托斯），第 238–238 页，IEEE 计算机协会，2020 年 5 月。[13] K. Guo、S. Zeng、J. Yu、Y. Wang 和 H. Yang，“基于 FPGA 的神经网络加速器综述”，2018 年。

[14] Z. Wang、Z. Jiang、Z. Wang、X. Tang、C. Liu、S. Yin 和 Y. Hu，“为集成式 CPU/GPU 异构平台实现延迟感知数据初始化”，*IEEE* 集成电路与系统计算机辅助设计汇刊，第 39 卷，2020 年 11 月，第 3433–3444 页。[15] C. Zhu、K. Huang、S. Yang、Z. Zhu、H. Zhang 和 H. Shen，“一种适用于 FPGA 上结构化稀疏卷积神经网络的高效硬件加速器”，*IEEE* 超大规模集成电路系统汇刊，第 PP 卷，第 1–13 页，2020 年 7 月。[16] J. Qiu、J. Wang、S. Yao、K. Guo、B. Li、E. Zhou、J. Yu、T. Tang、N. Xu、S. Song、Y. Wang 和 H. Yang，“借助嵌入式 FPGA 平台深入实现卷积神经网络”，载于*2016* 年 *ACM/SIGDA* 国际现场可编程门阵列研讨会论文集，FPGA ’16，（美国纽约州纽约市），第 26–35 页，计算机协会，2016 年。[17] K. Nan、S. Liu、J. Du 和 H. Liu，“移动平台深度模型压缩综述”，《清华大学学报（英文版）》，第 24 卷，第 6 期，第 677–693 页，2019 年。[18] J. Nunez-Yanez，“面向异构多精度神经网络的稀疏与稠密矩阵乘法硬件”，*Array*，第 12 卷，文章编号 100101，2021 年。[19] P. Jawandhiya，“机器学习硬件设计”，国际人工智能与应用期刊，第 9 卷，第 63–84 页，2018 年 1 月。[20] S. Fox、J. Faraone、D. Boland、K. Vissers 和 P. H. Leong，“使用 FPGA 以低精度和高准确率训练深度神经网络”，载于 *2019* 年国际现场可编程技术会议（*ICFPT*），第 1–9 页，2019 年。[21] https://docs.xilinx.com/v/u/en-US/xapp1231-partial-reconfig-hw-accelerator-vivado。访问日期：2022-04-1。



Jose Nunez-Yanez Jose Nunez-Yanez 博士是林雪平大学高能效自适应硬件架构领域的教授。此前，他曾任布里斯托大学副教授。他在拉夫堡大学获得基于硬件的并行数据压缩博士学位。他的主要专长是设计用于信号处理的可重构架构，重点关注运行时自适应、并行性和能效。2006—2007 年，他在 STM 担任玛丽·居里研究员

2011年，他在剑桥的 ARM Ltd 担任英国皇家学会研究员。2020—2022年，他在森萨塔科技担任英国皇家学会工业研究员。



Andr´esOtero 于2007年以优异成绩毕业于维戈大学，获得电信工程理学硕士学位。他于2009年和2014年分别获得马德里理工大学（UPM）工业电子学研究硕士学位和博士学位。目前，他是马德里理工大学电子学助理教授，同时也是工业电子中心（CEI）的研究员。他目前的研究兴趣集中于嵌入式系统设计、重

现场可编程门阵列上的可配置系统、可进化硬件和嵌入式机器学习。近年来，他参与了这些领域的不同研究项目，并在国际会议和期刊上发表了 40 多篇论文。



Eduardo de la Torre 是马德里理工大学（UPM，西班牙）的电子学副教授，在工业电子中心开展研究。他于2000年获得马德里理工大学电气工程博士学位。其主要专长包括现场可编程门阵列设计、嵌入式系统设计、硬件加速、信号处理，以及数字系统的部分和动态重配置。他参与了 40 多个项目，其中 11 个由欧盟资助；总计参与了 9 个获资助的

与可重构系统相关的项目。