将发表于 ASPLOS'23。本文为草稿。

Flexagon：用于高效 DNN 处理的多数据流稀疏-稀疏矩阵乘法加速器

Francisco Muñoz-Martínez* RaveeshGarg† José L.Abellán*Manuel E. Acacio* Tushar Krishna† Michael Pellauer‡

*穆尔西亚大学，†佐治亚理工学院，‡英伟达

联系作者：francisco.munoz2@um.es

arXiv:2301.10852v1 [cs.AR] 25 Jan 2023

摘要

稀疏性正成为现代深度神经网络模型的一大发展趋势。现有的稀疏-稀疏矩阵乘法（*SpMSpM*）加速器都针对某一种特定的稀疏矩阵乘稀疏矩阵数据流（即内积、外积或 *Gustavson*），而这种选择决定了它们的整体效率。我们证明，这种静态选择必然导致动态方案无法达到最优。这是因为不同的稀疏-稀疏矩阵乘法内核具有不同的特征（即维度、稀疏模式和稀疏度），因此不同的数据集更适合采用不同的数据流。

在这项工作中，我们提出了 *Flexagon*，这是首个可重构的稀疏矩阵乘稀疏矩阵加速器，能够针对不同情况采用最匹配的数据流执行稀疏矩阵乘稀疏矩阵计算。*Flexagon* 加速器基于一种新颖的合并归约网络（*MRN*），将归约与合并统一在同一硬件结构中，从而提高效率。此外，*Flexagon* 还采用三级存储层次结构，专门针对输入和输出压缩矩阵不同的访问特性进行设计。通过对来自多个应用领域的当代深度神经网络模型进行详尽的周期级仿真，我们表明，与当前最先进的类 *SIGMA*、类 *Sparch* 和类 *GAMMA* 加速器相比，*Flexagon* 的平均性能分别提升了 *4.59*×、*1.71*× 和 *1.35*×（按平均性能/面积效率衡量，分别提升 *265%*、*67%* 和 *18%*）。

1. 引言

张量中的稀疏性是现代深度神经网络工作负载中新兴的趋势 [30, 21, 23]。这些工作负载具有多样的稀疏率，范围从 0.04% 到 90%，并应用于各种应用程序，涵盖从个性化推荐 [23] 到自然语言处理 [3]。权重中的稀疏性源于剪枝 [11]，而激活值内部的稀疏性则源于 ReLU 等非线性函数。因此，通过直接实现稀疏矩阵乘法（SpMSpM）来利用稀疏性的优势，已成为定制化深度神经网络加速器的重要目标 [5, 12, 35, 25, 34, 31, 26]。

这些加速器最常见的稀疏性利用方式，是采用位图、压缩稀疏行和压缩稀疏列等压缩格式，仅存储并运算（乘法和累加）非零值。这样可以显著降低内存占用和运算次数，进而大幅节省能耗。然而，这些加速器在硬件实现和所采用的数据流方面差异很大。根据计算循环顺序，这些加速器使用的数据流大致可分为内积 (内积)、外积 (外积)和按行乘积，后者通常称为Gustavson (Gust) [9]。

加速器	架构特征	IP	OP	Gust
张量处理器 [15]	密集脉动阵列	N/A	N/A	N/A
SIGMA [5]	可配置归约树	✓	✗	✗
ExTensor [12]	交集单元	✓	✗	✗
MatRaptor [31]	合并器	✗	✗	✓
GAMMA [34]	纤维缓存，合并器	✗	✗	✓
OuterSPACE [25]	合并器	✗	✓	✗
SpArch [35]	矩阵凝聚器，合并器	✗	✓	✗
Flexagon （我们的方法）	灵活合并/归约树形与存储控制器	✓	✓	✓

表 1：Flexagon 与先前稀疏深度神经网络加速器在所支持的数据流方面的比较。IP=内积，OP=外积，Gust=Gustavson（按行乘积）。

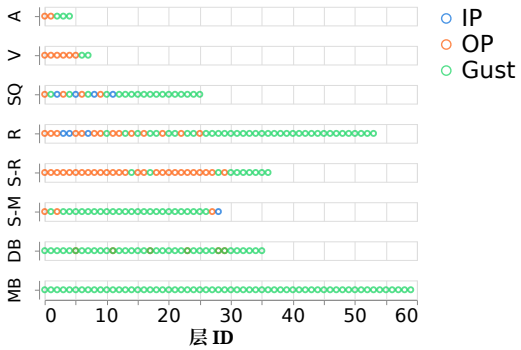


图 1：在各深度神经网络模型中逐层获得最佳性能的数据流（见表 2，其中包含这些模型的稀疏率）。IP=内积，OP=外积 和 Gust=*Gustavson*。

表1展示了现有稀疏加速器及其支持的数据流。尽管 SIGMA [5], Sparch [35] 和 GAMMA [34] 等最先进的稀疏加速器分别针对固定数据流（内积、外积 和 Gust）进行了优化，但本文提出一个重要观察：

最优数据流会因深度神经网络模型而异，甚至在同一深度神经网络模型中，也会因层而异，因此当代固定数据流加速器无法良好地适应变化，以最大限度地提升深度神经网络应用性能。

为了佐证我们的观察，图1展示了在执行来自MLPerf基准测试套件 [30] 的 8 个完整深度神经网络模型以及一些额外模型（详情见表 2）时，每层性能最佳的数据流。请注意，我们考察了来自不同领域、规模各异且稀疏率不同的异构模型。对于 *MobileBERT*，我们仅展示前 60 层，这些层占总层数的 20%。为建模这三种数据流，我们在 64乘法器的类 SIGMA、类 Sparch 和类 GAMMA 架构上执行了测试（更多详情见第 4 节）。自然语言处理模型 *DistilBERT* 和 *MobileBERT* 明显倾向于Gust。另一方面，极度稀疏的模型，如*S-R* 和 *V*，分别在 73% 和 75% 的层中受益于 外积 。其余深度神经网络模型在不同层之间表现出较大的差异；鉴于各层的特征不同，最高效的数据流也会随之变化。这表明不存在一种适用于所有情况的数据流，因此可以通过动态调整架构组件，使其采用最合适的数据流，从而提高效率。

支持稠密深度神经网络采用灵活数据流的价值已得到广泛探讨 [18, 4, 19, 2]。然而，由于这些加速器处理稀疏性的方式各不相同，要支持面向稀疏工作负载的灵活数据流加速，挑战要大得多。例如，SIGMA 中实现的内积数据流 [5]采用一种称为 FAN 的归约网络，可一次性归约生成的部分和，同时还具备执行交集以完成稀疏点积的能力。相反，外积和Gust 数据流由 Sparch 等加速器实现 [35]和 GAMMA [34]产生的是部分和而非完整和，因此需要合并非零部分和，并为此使用合并树。若采用彼此独立的硬件模块分别执行归约和合并，朴素的实现方式将带来显著的面积开销（见第5.3节）。

为了高效支持不同的稀疏矩阵乘稀疏矩阵工作负载，以运行现代稀疏深度神经网络，我们提出了*Flexagon*，据我们所知，这是首个可重构的稀疏同构深度神经网络加速器，可针对每个深度神经网络层动态调整，以执行最适合的稀疏矩阵乘稀疏矩阵数据流。Flexagon 配备了一种新颖的统一式合并归约网络（*MRN*），既支持点积归约，也支持部分和合并。我们提出了一种基于树的拓扑结构，其中的节点可配置为累加器或比较器，详见第3节。Flexagon 还采用了一种新型 L1 片上存储器组织，由三种定制存储结构组成，能够捕获每种数据流的内存访问模式。第一种存储结构是简单的只读先进先出队列，专为三种数据流在某些阶段发生的顺序访问而设计。第二种是低功耗缓存，用于支持由主要由Gust 数据流引起的随机访问。最后，一种名为 PSRAM 的定制存储结构专为存储和读取部分和而设计，这对于外积 和Gust 数据流都至关重要。这些存储结构使我们能够以极小的面积与功耗开销支持全部三种数据流。此外，我们的加速器还能避免硬件在层与层之间进行代价高昂的显式压缩格式转换（即从 CSR 转换为 CSC，反之亦然） [27]，因为对于特定的压缩格式，可以轻松切换到最合适的数据流（详见第3节）。

我们总结了主要贡献：

(1) 我们证明，现代稀疏深度神经网络层中的每种稀疏矩阵乘稀疏矩阵运算，都会因矩阵维度和稀疏模式不同而呈现不同的内存访问模式。因此，使特定稀疏矩阵乘稀疏矩阵运算性能最大化的数据流不仅会因深度神经网络模型而异，也会在同一深度神经网络模型的不同层之间发生变化。

(2) 我们提出了一种新的层间数据流机制，无需显式硬件模块即可实现压缩格式转换。

(3) 我们设计了 Flexagon，其核心是一种新颖的网络拓扑（称为 MRN），首次支持三种数据流；此外，我们还设计了一种新的 L1片上存储器组织方式，可有效捕获每种数据流在输入、输出和部分和方面呈现的内存访问模式。

(4) 我们通过对来自不同应用领域的若干当代深度神经网络模型进行周期级仿真，并对其核心组件进行 RTL实现，对 Flexagon 进行了全面评估。结果表明，与最先进的类 SIGMA、类 Sparch 和类GAMMA加速器相比，Flexagon 的平均性能分别提升了 4.59× （范围为 2.09× 至 7.41×）、1.71× （范围为 1.04× 至 4.87×）和 1.35×（范围为 1× 至 2.13×）；在平均性能/面积效率方面，分别提升了 265%、67% 和 18%。

2. 背景

2.1. 压缩格式

遵循 ExTensor 中使用的相同分类法 [12]，，Sp M-SpM 运算计算以下运算：$C_{M,N} = A_{M,K} \times B_{K,N}$其中这三个矩阵都是二维张量。由于这些矩阵通常是稀疏的（见表2），因此会对其进行压缩，以编码非零值，同时保持计算不变（无损压缩） [17]。在我们的工作中，我们聚焦于广泛使用的非结构化压缩格式 CSR 和 CSC。采用 CSR格式编码的矩阵使用三个一维张量，以行主序数据布局存储非零值：数据向量用于表示非零值，行指针向量用于存储每行在数据向量中的起始索引位置，列索引向量用于存储每个非零值所在的列。类似地，CSC 使用列主序数据布局：一个数据向量、

DNN	Appl	nl	平均速度A	平均速度B	平均成本A	平均成本B	最小成本A	最小成本B	最大成本A	{最大成本B	周期(10^6) 中央处理器
AlexNet (A)	CV	7	70	48	0.56	13.6	0.02	0.18	1.01	63.41	3804
SqueezeNet (S)	CV	26	70	31	0.05	1.54	0.001	0.02	0.58	26.6	2751
VGG-16 (V)	CV	8	90	80	0.55	2.90	0.02	0.15	10.42	0.90	6012
ResNet-50 (R)	CV	54	89	52	0.19	1.30	0.001	0.007	1.0	26.64	4185
SSD-ResNet (S-R)	OR	37	89	49	0.12	3.60	0.003	0.003	10.1	0.50	6429
SSD-MobileNet (S-M)	OR	29	74	35	0.16	0.31	0.002	0.0004	1.0	1.65	5379
DistilBERT (DB)	NLP	36	50	0.04	2.25	0.35	1.12	0.23	4.5	0.94	5748
MobileBERT (MB)	NLP	316	50	11	0.10	0.07	0.03	0.003	0.125	0.01	4893

表2：本文使用的深度神经网络模型。应用程序=应用领域。计算机视觉=计算机视觉，OR=目标识别，自然语言处理=自然语言处理，网络层=层数，AvSp{A,B}=矩阵{A,B}的平均稀疏性（百分比），AvCs{A,B}=矩阵{A,B}的平均压缩大小（MiB）。MinCs{A,B}=矩阵{A,B}的最小压缩大小（MiB）。MaxCs{A,B}=矩阵{A,B}的最大压缩大小（MiB）。周期(10^6) 中央处理器 =在中央处理器系统中使用英特尔数学核心函数库运行基准测试后获得的周期数。

一个列指针向量，用于存储列起始位置的索引，以及一个行索引向量，用于存储每个非零数据值的行索引。请注意，CSR 和 CSC 使用相同的压缩方法，因此可以视为单一压缩格式。这一点很重要，因为加速器可以使用相同的控制逻辑来处理二者。这有助于实现加速器中的控制逻辑（详情见第3.5节）。

与以往研究（例如 [34]）一样，我们将使用纤维一词来表示每个压缩行或列。每个纤维都包含一个按坐标排序的二元组（坐标，值）列表。我们使用元素 一词来指代纤维中的一个二元组。

2.2. 稀疏矩阵-稀疏矩阵乘法数据流

稀疏矩阵乘稀疏矩阵运算基于三重嵌套循环，该循环遍历A和B的独立维度*M* 和*N*，并对它们共有的维度*K*进行协同迭代。根据循环嵌套中协同迭代所在的层级，已为稀疏矩阵乘稀疏矩阵计算确定了三种不同的数据流：内积 （在最内层循环进行协同迭代）、外积 （在最外层循环进行协同迭代）和Gust （在中间层循环进行协同迭代）。此外，这些数据流会导致

六种可能的变体取决于独立维度（*M* 和*N*）在每种变体中的排列方式（每种数据流对应两种变体）。请注意，每种变体都更偏向于其中一个维度（最外层维度）的驻留性，而非另一个维度的驻留性。因此，我们根据计算是 （M） 驻留还是*M*-驻留，或是 （N） 驻留，即*N*-驻留，来区分各个变体。图2 展示了由此产生的六种数据流变体。每种数据流定义了元素在执行期间的流动方式，以及数据复用的机会。表3给出了每种方法的详细分类，概括如下：

内积（IP）：每次生成一个完整的求和结果，不需要合并部分和。这需要硬件求交单元来对齐有效输入。

外积（OP）：每次消耗一个输入标量，并生成多个部分和。这不需要求交，但需要合并硬件，并且可能产生额外的存储器流量。

Gustavson（Gust）：每次消费一个输入，但仅用于将部分和生成到当前纤维中。这种方

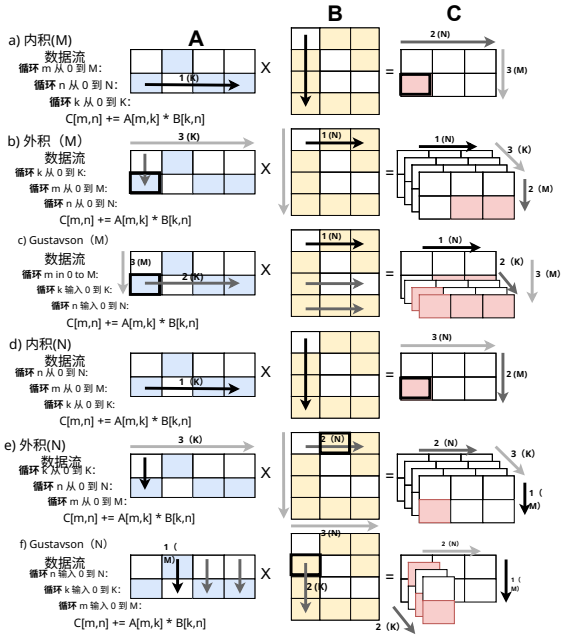


图2：矩阵乘法的数据流组合。为简洁起见，图中展示的是未压缩（稠密）矩阵。

式使得交集能够以主从方式完成，其中驻留张量的有效坐标会检索另一个操作数的整个纤维。这需要合并硬件，但合并范围仅限于当前纤维，而非整个矩阵。

在本文余下部分的讲解中，为便于说明，我们将采用*M*驻留数据流，尽管*N*驻留数据流也同样适用。

3. Flexagon 设计

图3a 展示了 Flexagon 加速器架构的高层概览。如图所示，Flexagon 由一组乘法器、加法器和比较器，以及三个片上静态随机存取存储器模块组成；这些模块专为存储三种稀疏矩阵乘稀疏矩阵数据流所需的矩阵A、B 和C 而设计。此外，为了实现最高的灵活性，所有片上组件都通过一个通用的三层可重构片上网络（NoC）互连，该网络由分配网络（DN）、乘法器网络（MN）以及

数据流	非正式名称	驻留张量	驻留纤维	流式处理张量	A 格式	B 格式	C 格式	交集	合并
MNK	内积（M）	C	A	B	CSR	CSC	CSR	标量 A 与标量 B	N/A
KMN	外积（M）	A	B	C	CSC	CSR	CSR	N/A	标量
MKN	Gustavson（M）	A	C	B	CSR	CSR	CSR	标量 A 与纤维 B	纤维(M)
NMK	内积(N)	C	B	A	CSR	CSC	CSC	标量 B 与标量 A	N/A
KNM	外积(N)	B	A	C	CSC	CSR	CSC	N/A	标量
NKM	Gustavson(N)	B	C	A	CSC	CSC	CSC	标量 B 与纤维 A	纤维(N)

表 3：数据流属性分类。遍历顺序按循环顺序从最外层到最内层排列。

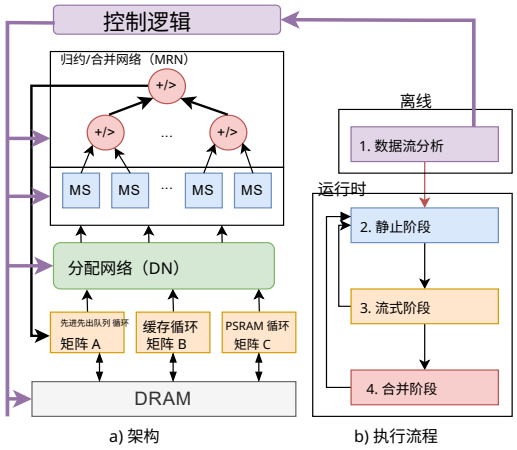


图 3：Flexagon 高层概览。

合并归约网络（MRN），其设计灵感源自人工智能加速器片上通信流的分类法 [19]组成。这些组件由控制单元控制，而控制单元在执行前由映射器/编译器进行配置。

Flexagon 的执行阶段如图 3b 所示。整个过程从数据流分析（阶段 1）开始，该分析以离线方式进行。在此，映射器/编译器会检查待执行的稀疏矩阵乘稀疏矩阵运算的特征（即矩阵维度和稀疏模式），并从第2节所述的六种可用数据流中选择最适合该运算的一种，同时生成分块方案以及用于配置加速器其余阶段运行的具体信号值。

接下来的三个阶段根据生成的信号在运行时执行，并根据执行分块的数量重复多次。第一个运行时阶段称为**驻留阶段**（阶段 2），负责传送数据，使其驻留在乘法器中，从而减少代价高昂的内存访问。根据第2 节对 *M*驻留数据流的描述，这些驻留数据属于矩阵A，而矩阵B 则在**流式阶段**（阶段 3）中以流式方式传送。对于*N*驻留数据流，顺序则相反。这两个阶段适用于这三种数据流。**合并阶段**（阶段 4）仅对外积 和Gust 数据流有必要；该阶段负责合并流式阶段中先前生成的部分和纤维。在内积 数据流中，由于无需合并，因此会跳过此阶段。

在这项工作中，我们重点关注加速器的设计，以及这三个阶段的运行方式，以便在同一硬件底层平台上支持六种可能的数据流（三种稀疏矩阵-稀疏矩阵乘法数据流，每种各有两种变体，即 M 驻留或 N 驻留）。我们将数据流分析、分块选择以及为加速器生成配置文件所需工具的研究（图中离线部分的第 1 阶段3b）留待未来工作。

3.1. 片上网络

Flexagon 的一项主要创新在于，它能够通过适当配置，在使用相同硬件底层平台的情况下支持第 2 节所述的六种数据流。为此，该加速器配备了三级可配置片上网络，能够适应每种数据流的通信特征。接下来，我们将详细介绍各个子网络：

分发网络 (DN)：该模块用于将数据从 SRAM 结构传送到乘法器。为了满足三种稀疏矩阵乘稀疏矩阵数据流所需的高度灵活性，DN 必须支持单播、多播和广播数据传送。为此，同时确保高能效，我们采用了与 SIGMA [5] 等先前设计类似的贝内斯网络。该网络是一种具有 $2 \times log(N) + 1$ 个层级、每层包含 N 个微型 2×2 开关的 N 输入、N 输出无阻塞拓扑。

合并归约网络(MRN)：以往的设计，如 MAERI [19] 或 SIGMA [5]，采用 ART 或 FAN 等专用树形归约网络（RN），以支持多个部分和簇的非阻塞归约。这些 RN 为<style id='5'>内积 数据流提供了很高的灵活性，因为其用途是归约一个部分和簇。对于外积 和Gust 数据流，其他工作（如 [34, 35] ）则采用树形拓扑，在部分和生成后执行合并操作。在我们的设计中，我们首次将这一概念统一起来，设计出一种既能归约又能合并部分和的合并归约网络。图4a 展示了一个 16 路 MRN 的微架构概览。可以看到，与以往设计类似，我们也采用增强树形拓扑，因为这是执行归约与合并操作最快的方式。与以往设计不同的是，MRN 拓扑在节点中增设了比较器和交换逻辑，使其能够切换操作模式（见图4b）。这样便能同时执行这两种操作，同时将面积与功耗开销保持在较低水平（详见第5.3节）；此外，正如后文所述，这也使其能够直接支持三种稀疏矩阵乘稀疏矩阵数据流。除此之外，我们在节点之间采用双链路连接，使 MRN 不仅能够传递数据值，还能传递外积 和Gust 数据流所需的坐标。

配置的选择由映射器/编译器完成；它会生成控制信号并馈入加速器的配置逻辑模块（图3a中的控制逻辑），后者再将相应信号路由至各节点，根据数据流和层维度配置其操作模式。

乘法器网络（MN）：与 MAERI 等其他设计类似，该网络由彼此独立的乘法器组成，可工作于两种不同模式：i）乘法器模式：单元执行乘法，并将结果发送至 MRN。当配置内积数据流时，整个执行过程中均采用此模式；配置外积或Gust 数据流时，则在流式阶段采用此模式；ii）转发模式：乘法器将输入（通常为部分和）直接转发至 MRN。正如接下来示例中将进一步说明的，在外积和Gust 数据流中，此模式基本上是在合并阶段配置的。乘法器的微体系结构如图4c 所示。

3.2. 示例演示

接下来，我们将说明 Flexagon 在运行三种数据流以计算图2中的矩阵A和B的乘积时如何工作，并考虑前文解释的运行时阶段。为便于说明，我们假设采用IP(M)、OP(M) 和Gust(M) 数据流。请注意，只需交换矩阵IP(N)、OP(N) 和Gust(N) 数据流即可用同样方式执行。为便于讲解，我们假设使用一个简单的四乘法器加速器，并逐步介绍三个子网络的工作过程。在说明中，我们会提及存储矩阵A 、B、C以及部分和所需的片上 SRAM 模块（见图3b中的黄色方框）。第3.4节将深入介绍这些存储结构。

3.2.1. 内积数据流示例。图5 展示了IP(M) 数据流。图中，我们用“*”表示需要由树中的加法器归约、以生成矩阵C最终值的部分和。

静止阶段：首先，在静止阶段，控制器将矩阵A 尽可能多的纤维（即A的行）映射到乘法器，并从矩阵A专用的、称为先进先出队列的 SRAM 结构中依次读取所有元素。每个乘法器簇将执行点积运算。

流式阶段：将A的纤维装入乘法器后，控制器会将矩阵B的每个纤维（即每一列）多播到乘法器网络中配置好的簇。为此，控制器利用B 纤维中每个元素的行坐标，检测其是否与A纤维中的列坐标相交。若相交，则将该值发送给相应的乘法器。

3.2.2. 外积数据流示例。图6展示了与之前相同的示例，但现在假设采用OP(M) 数据流。我们还展示了为C 定制的 SRAM结构，称为 PSRAM，用于存储矩阵C的部分和。正如我们将在第3.4节中解释的那样，该结构存储由元素（坐标、值）组成的块。

静止阶段：在静止阶段，矩阵 A 的纤维（即 A 的列）按照 CSC 压缩格式依次传送给乘法器。在我们的具体示例中，四个乘法器分别存储元素 $A^{1,0}$、$A^{0,1}$、$A^{1,2}$、和 $A^{1,3}$。

流式阶段：在流式阶段，每个乘法器将元素 $A_{m,k}$ 驻留其中（m 属于 range [0,M)，且 k 属于 range [0,K)），以便对非零元素 $B_{k,0:N-1}$ 进行线性组合，从而生成一个部分和纤维，其中所有元素都具有相同的行 (m) 和特定的 k 次迭代（即这些元素所属的部分矩阵）。对于同一次 k 次迭代，连续的乘法器为不同行生成部分和时，无需将这些部分和合并。因此，生成的部分和必须发送到静态随机存取存储器结构中，以便在第三阶段进行合并。此外，由于多行可以并行处理，PSRAM 的集合按行索引。另外，由于矩阵 A 中的非零元素数量无法预先得知，矩阵 A 中的多个纤维可能会适配到单次迭代中，导致同一行、但属于不同 k 次迭代的多个部分输出可能并行处理。由于运行时无法得知特定行和特定 k 次迭代对应的部分和数量，我们必须在 PSRAM 中分配静态空间，以存储可能并行处理且位于同一行的不同 k 次迭代所产生的部分和。为此，我们将 PSRAM 中的每一行划分为多个块，每个块包含一个有效位，用于指示数据是否有效；一个 k 值，用于指示该组部分和所属的 k 次迭代；以及数据块。这样，每个块在任一时刻都可以存储特定行中属于不同 k 次迭代的部分和。因此，如果特定迭代的部分和数量超过块大小，则可以使用该行中的另一个块，即使下一个块已被另一次 k 次迭代占用。PSRAM 的组织与操作详情见第3.4节。

在图 6的示例中，我们可以看到流式阶段的三个步骤。第一步，控制器将四个纤维（沿 K维度）的第一个元素发送到对应的乘法器。例如，驻留元素 $A^{1,0}$ 的第一个乘法器接收行纤维（即迭代 k）0 的第一个元素。第二步，每个乘法器生成一个部分和（以符号 *表示），它是沿K维度生成的 4 个纤维的第一个元素。随后，这些部分和存储在 PSRAM 中。第一个部分和 $*C^{1,1}$ 被分配到集合 1，因为它根据行坐标进行索引。使用集合可以让我们并行执行多行。接着，由于第一行空闲，部分和被存储在那里，同时启用有效位，并表明该元素属于 $K0$。将行划分为块，可以保存特定行中对应不同 K 的部分和。第二个部分和 $*C^{0,0}$ 被分配到集合 0（其行坐标），由于该集合中的第一行空闲，缓存便启用有效位并

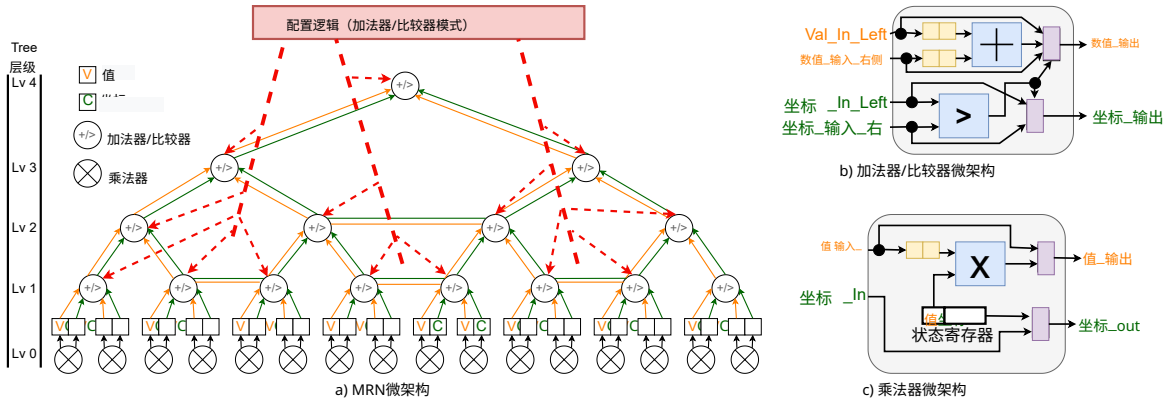


图 4：a) MRN 拓扑。b) μMRN 节点的架构（加法器/比较器节点）。c) μ乘法器的架构。

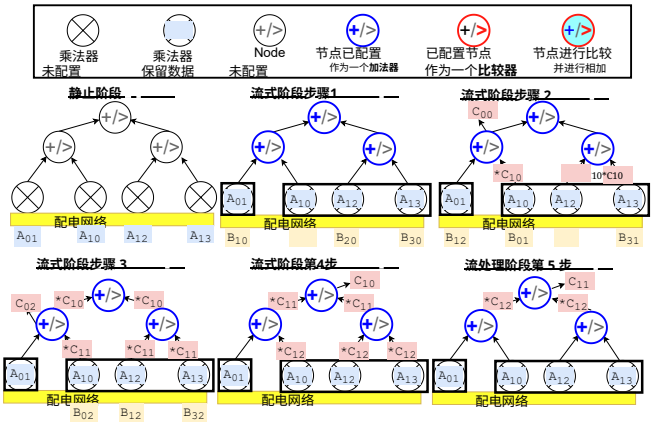


图 5：Flexagon 使用内积(M)数据流运行稀疏矩阵乘稀疏矩阵的示例。“*”表示部分和。

将该行标记为 *K1*。最后两个元素具有相同的坐标（即 s $*C^{1,0}$），但属于不同的部分矩阵（*K2*和*K3*）。这两个元素进入 PSRAM 中的同一个集合，但位于不同的行中，每行都标记有其迭代 k （即 *K2*和*K3*）。这样，在合并阶段就能按正确顺序定位部分和纤维。

在第 3 步中，按照相同的执行方案，生成四个纤维的第二个元素。为简洁起见，我们不展示如何生成最长部分和纤维（即纤维 K3）的最后一个元素，而是直接展示开始合并阶段（合并阶段第 1 步）前 PSRAM 中的内容。从合并阶段第 1 步的 PSRAM 图中可以看到，该元素已存储在第一个集合的最后一行中，因为第三行已经存满。

合并阶段：合并阶段按行进行。对于每一行，控制器从 PSRAM 中取指不同 k 次迭代纤维的元素。这些元素存储在不同的块中，并可通过其标签识别；控制器取出这些元素并将其发送到 MRN 进行合并。MRN 中的每个单元都会比较列坐标（即 N 维）。如果坐标相同，则累加这些元素的值；否则，节点会将坐标最小的元素沿树向上传递。图 6 的最后两行展示了 8 个合并步骤。

前 4 个步骤（合并阶段第 1 步至第 4 步）合并第一行。第二行中有 3 个待合并的部分和纤维。步骤 5 中，三个纤维（*K0*、*K2* 和*K3*）的首个元素被发送到 MRN。步骤 6 中，部分和 $*C^{1,1}$ 和 $*C^{1,0}$ 比较它们的列坐标。由于二者坐标不同，且元素 $*C^{1,0}$ 的列坐标最小，因此该元素首先沿树向上发送到 MRN。对于纤维中的其余元素，也以流水线方式执行相同的过程，直到所有部分和都合并到单个纤维中并发送到动态随机存取存储器。如果一行中的纤维数量大于乘法器数量（即树中的叶节点数），控制器就需要执行多遍，才能完成最终合并。

3.2.3. Gustavson 数据流示例。最后，对于相同的示例矩阵，图 7 展示了当选择Gust (M) 数据流时 Flexagon 的执行过程。类似地，此处的操作分为三个截然不同的阶段。

静止阶段：首先，在静止阶段，尽可能多的A 纤维（即矩阵A中的行）会以空间和顺序方式映射到乘法器中。随后，乘法器会保留两个簇，每个簇负责计算不同输出行的部分和（即示例中的第 0 行和第 1 行）。

流式阶段：在流式阶段，对于每个乘法器，存储器控制器都会取指并传送B 的纤维（即B的行），该纤维对应于与乘法器中映射的A 元素相关联的列坐标（即k迭代）。每个乘法器都会生成一个部分输出纤维，并将其与分配给A同一纤维的其他乘法器所生成的其余部分输出纤维合并。图7展示了这一生成过程的示例。这里展示了 6 个流式步骤。第一个乘法器将矩阵A 中唯一的元素（$A^{0,1}$）保持驻留，因此它接收按列 1 编索引的B纤维（即第 1 行）。第二、第三和第四个乘法器分别将元素$A^{1,0}$、$A^{1,2}$ 和$A^{1,3}$ 保持驻留，因此它们分别接收B 的纤维 0、2 和 3。前 3 个步骤展示了B 纤维中的元素如何逐周期传送。

合并阶段：与外积 数据流类似，合并阶段同时执行累加和合并操作，

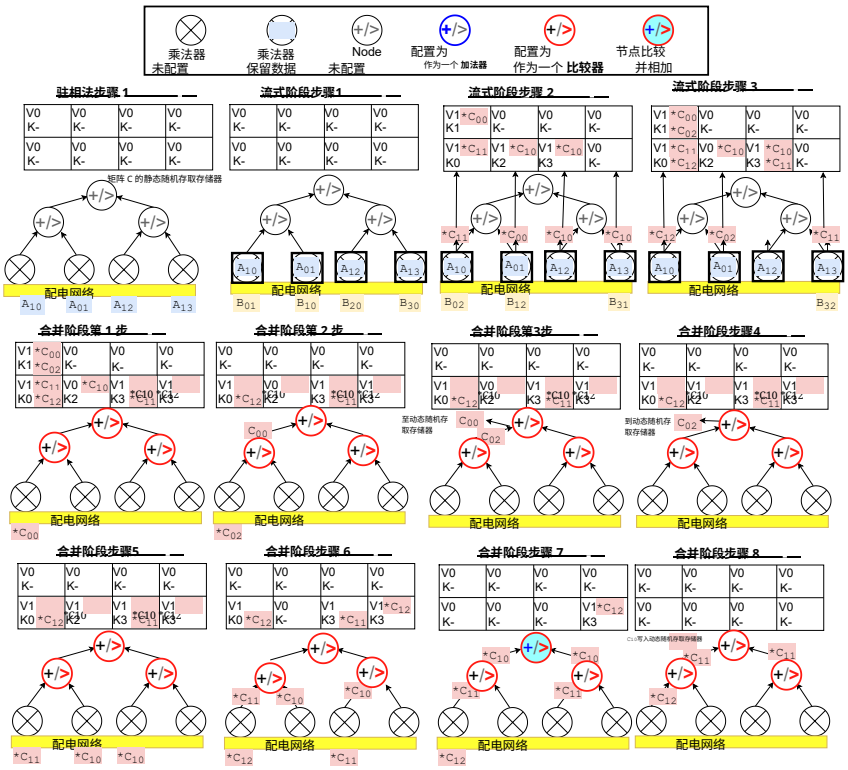


图 6：Flexagon 使用 外积（M）数据流执行稀疏矩阵乘稀疏矩阵的示例。“*”表示加速器生成的输出是部分和，而非最终输出。PSRAM 中的“*V*”表示有效位，“*K*”表示针对特定块标记的第*k*次迭代。

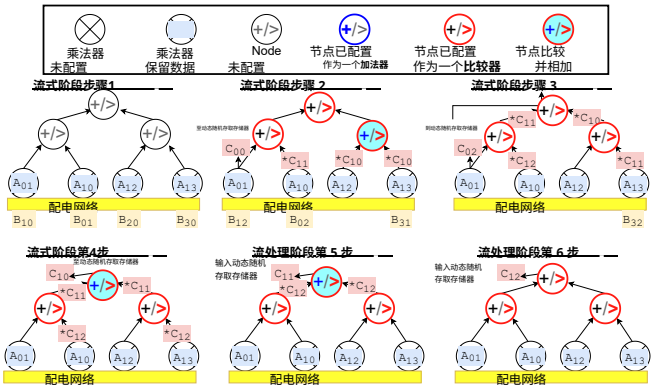


图 7：Flexagon 使用Gustavson（M）数据流执行稀疏矩阵乘稀疏矩阵的示例。“*”表示加速器产生的输出是部分和，而非最终输出。

如果某个节点中的元素（即其值）的列坐标相同，就将这些元素累加；否则，发送列坐标值最小的元素。另一方面，在Gust 数据流中，部分和生成后即可立即合并，因为乘法器簇始终为同一行生成纤维，但对应不同的*k*迭代。当A 中的元素数量足以放入一个乘法器簇时，该簇生成的输出纤维就是最终纤维，输出可以直接发送到动态随机存取存储器，无需存储在静态随机存取存储器中。否则，当元素数量

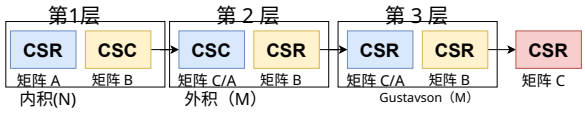


图 8：通过运行内积、外积 和Gustavson 数据流的组合来执行三个深度神经网络层的示例。

如果A 中的数量超过乘法器的数量，则输出纤维将是部分纤维，因为需要进行多次迭代，因此该纤维需要存储在 PSRAM 中，这与外积数据流中的情况类似。

3.3. 层间数据流的组合

如表3 所示，*M*驻留数据流以 CSR 格式输出元素，而*N*驻留数据流以 CSC 格式输出元素。Flexagon 支持这六种数据流，并利用这一观察结果，在无需昂贵的显式硬件格式转换的情况下，恰当地执行所有可能的深度神经网络层序列；它也是首个支持无需显式转换即可生成压缩输出的方案。图8 展示了一个由三层组成的深度神经网络示例，说明了这一特性。第一层和第二层分别配置为执行内积和外积。由于第二层需要 CSC 格式的激活值，因此第一层采用内积（N）。假设权重以两种格式离线存储。第二层在采用 M 驻留时生成 CSR 格式的矩阵

	IP(M)	*OP(M)*	*Gust(M)*	*IP(N)*	*OP(N)*	*Gust(N)*
IP(M)	✓	EC	✓	✓	EC	EC
OP(M)	✓	EC	✓	✓	EC	EC
Gust(M)	✓	EC	✓	✓	EC	EC
IP(N)	EC	✓	EC	EC	✓	✓
OP(N)	EC	✓	EC	EC	✓	✓
Gust(N)	EC	✓	EC	EC	✓	✓

表4：无需显式格式转换（EC）即可进行的数据流转换。不同的行表示第一层的不同输出，不同的列表示第二层对应的输入。

格式。因此，它可以选择内积或 Gustavson(M)。

表 4 展示了每种数据流组合中无需显式格式转换（绿色勾号）和需要显式格式转换（显式格式转换 或 EC）的转换。这些组合可供映射器/编译器利用，以生成最佳数据流序列，从而在特定深度神经网络执行中实现最佳性能和能效。

3.4. 存储器组织

为了涵盖所有数据流，我们专门设计了定制的 L1存储器层级，以适应三种数据流中共有及各异的模式。图9 展示了该 L1 存储器层级的示意设计。我们为每种存储器结构分别采用独立的存储器结构和不同的缓冲区惯用模式，以实现数据在各结构之间的移动。为此，每种存储器结构均由两个控制器操作：与动态随机存取存储器交互的**图块填充器**，以及与加速器的数据通路（即乘法器）交互的**图块读取器**。接下来，我们将详细介绍每种存储器结构：

驻留矩阵的存储器结构（先进先出队列）：对于三种数据流，驻留矩阵的元素始终只读取一次，并按顺序读取，因为它们驻留在乘法器中。为隐藏访问延迟，我们实现了一个 512 字节的只读先进先出队列。为节省带宽并降低复杂度：(1) 存储器结构将驻留矩阵的DRAM位置保存在寄存器中，因此纤维会隐式地压入先进先出队列；(2) 读取和写入共用一个单端口。

流式矩阵的存储器结构（缓存）：流式矩阵呈现出更复杂多样的内存访问模式。在 内积中，每个驻留阶段（即每次迭代）都会流式处理整个矩阵。换句话说，每次重新加载矩阵时，都具有显著的空间局部性和时间局部性。在 外积 数据流中，流式矩阵的纤维只读取一次，并按顺序读取。在
Gust 数据流中，驻留矩阵的每个纤维都会收集F个流式矩阵的纤维，其中F 是驻留矩阵中该纤维的非零元素数量。这些元素通常散布在整个矩阵中，造成不规则且不可预测的存储器访问模式。为应对最坏情况下的Gust 数据流，我们将流式矩阵的存储器结构实现为传统的只读组相联

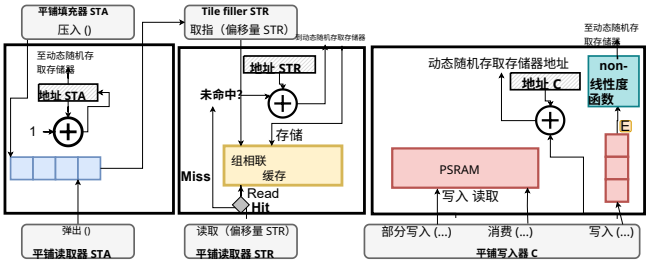


图 9：Flexagon 中的存储结构。

缓存。不过，我们将此缓存配置为在相对于流式矩阵起始位置的虚拟地址空间中运行，这使我们能够使用更短的内存地址，从而节省带宽并缩短标签长度。

矩阵 C 的存储器结构（PSRAM）：为了存储部分和，我们设计了一种名为*PSRAM* 的新型缓冲区惯用模式，它同时用于外积和Gust 数据流。图6 展示了这种存储器结构的工作方式，图9 展示了高层示意图，而图10 则深入介绍了细节。存储器被组织成多个集合，分别对应不同的行；每个集合又被划分为多个块，分别对应一行中不同的 K 维度。每个块都有一个有效位。此外，我们使用一个寄存器作为行标签，以保存分配给该行的列坐标（即k次迭代）。由于输出纤维的长度无法预先确定，它可能会占据同一行中的多条（且不连续的）行。这本质上是一种由k次迭代标记的路合并方案。该寄存器供该行使用，以确定某个输出纤维是否仍存放在 PSRAM 中。为了从同一集合中并行读取多个纤维（即合并特定的行或列），我们采用了一种跨集合内各行组织的多银行方案。最后，我们还加入了两个寄存器，用于记录该行中首个和末尾元素所在的字节位置。

部分写入(行, k, E)：此操作用于在 PSRAM 中放置一个元素。逻辑根据行参数索引该元素，然后并行搜索存储带有标识符k 的输出纤维的行。如果输出纤维存在（即k 标签匹配），PSRAM 会将新元素E 放入最后一行中最后一个可用位置（由元数据中的*Last* 寄存器指示）。如果该纤维不存在，逻辑会搜索第一个可用行，并将元素E 存储在该行的第一个位置，同时启用有效位并更新K、*First* 和*Last* 寄存器，以便在后续访问中继续存储同一K 标识符对应的元素。

消费(行, k)：部分输出纤维中的元素会暂时存放在 PSRAM 中。它们会被读取一次以供加速器使用，之后不再使用。这样，我们便可引入 **消费** 操作，该操作会从存储器结构中读取并删除特定元素。具体而言，控制器会逐行合并部分输出纤维。为此，控制器需要尽可能多地读取同一行的纤维，并对每条纤维使用 **消费** 操作，指示行和纤维 k 以

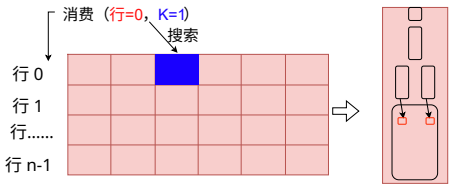


图 10：PSRAM 概览。

```
输入
- p_A：行指针向量 STA - i_A：列指针向量 STA - d_A：数据向量 STA
- p_B：STR 行指针向量 - i_B：STR 列指针向量 - d_B：STR 数据向量
- p_C：行指针向量 C - i_C：列指针向量 C - d_C：数据向量 C
Tile 填充器/读取器 STA
1: 循环（整数 i=0; i<p _A.size; i++) {
2: 循环（整数 j=p_A[i]; j<p_A[i+1]; j++）: 3:
mem. 压入 () / mem. 弹出 ()
瓦片填充器/读取器 STR
1: 整数 高 _L1=(古斯塔夫森算法||内积 _积)：p_A.大小？ 1
2: 循环 (int i=0; i<n_high_L1; i++): 3: int low
4: int high_L2=(古斯塔夫森算法) : p _A[i] ? 0
_L2=(古斯塔夫森算法) : p_A[i+1] ? p_B.size 5: 循环 (int j=low
6: 当前_指针=（古斯塔夫森算法）：接收(i_A[i]) ? 7: _L2; j<high _L2; j++):
整数 下界
_L3 = p _B[当前_指针]
8: 整数 上界_L3 = p_B[当前_指针+1]
9: 循环 (整数 k=下界_L3; k<高_L3; k++）:
10: 内存.取指[k] / 内存.读取[k]
瓦片写入器 C
1: 整型 rcv _elems=0
2: 当（pck=pending _元素()）:
3: 值=pck.data()
4: 行=pck.row()
5: 列=pck.col()
6: if(pck.is _部分 _求和()）:
7: 部分写入(行, 列, 值);
8: 否则：
9: d_c[rcv_元素]=值 10: i
_C[接收_元素]=列
11: 如果（新_行（行）？ 12: p_C[行]=
rcv_元素
```

图 11：平铺填充器 STA、平铺读取器 STA、Tile filler STR、平铺读取器 STR 和平铺写入器 C 的伪代码。我们将填充器和读取器合并在同一个文本框中。STA：驻留，STR：流式处理。

进行搜索。如果存在保存 k 纤维的有效行，存储器结构会从该纤维中读取下一个元素（由寄存器 *First* 指示），并将该寄存器加一，从而消费该元素。当 *First* 和 *Last* 寄存器存储相同值时，PSRAM 会检测到该行已被消费，并将有效位置 0，从而使该行失效。

写入（偏移量，E）：除了用于存储部分输出纤维的 PSRAM 外，我们还在存储器结构中增设了一个先进先出队列，用作写入缓冲区，以隐藏将最终输出纤维发送至动态随机存取存储器的延迟。该结构由**写入**操作使用。

3.5. 存储器控制器

为每种数据流与存储器结构的组合分别配置一个存储器控制器，在面积和功耗方面的代价会非常高，因为这需要 30 个逻辑模块来协调数据（*6* 种数据流 $\times$ *5* 存储器控制器）。在我们的设计中，我们统一了逻辑，每个控制器都可以根据各数据流的存储器访问模式进行配置。这样，如图 9所示，对于保持驻留的存储器结构（即图块填充器 STA 和图块读取器 STA），对于流式矩阵的存储器结构需要两个存储器控制器来协调（即图块填充器 STR 和图块读取器 STR），而对于 C 的存储器结构则需要一个控制器来协调（即图块写入器 C）。图11 展示了这些统一存储器控制器的代码。

4. 实验方法

仿真基础设施：为了对 Flexagon 进行详细评估，我们利用 STONNE 框架 [22][1]实现了一个周期级微架构模拟器，用于模拟加速器的所有片上组件。为了忠实地模拟整个存储层次结构（包括片外 HBM 2.0 DRAM），我们将模拟的加速器连接到结构仿真工具包 [14]。表 5 展示了我们为后续评估配置的架构的主要参数。我们将结果与三种最先进的加速器进行比较：类 SIGMA（作为内积加速器的示例）、类 Sparch（作为外积加速器的示例）和类 GAMMA（作为 Gust 加速器的示例）。

我们使用 *-like* 一词来表示 GAMMA-*like*、SIGMA-*like* 和 SpArch-*like*，以体现我们在模拟器中以公平且标准化的方式捕捉了它们最相关的特征（即其本质）。具体而言，我们关注数据流（这是影响加速器效率的关键部分）、DN、MN 和 RN 组件（它们决定加速器的规模和带宽），以及片上存储器结构（它们决定加速器将数据存储在处理单元附近的能力）。我们指出，这些设计均酌情获得了额外的片上面积。SIGMA、SpArch 和 GAMMA 的效率主要分别来自 FAN 归约网络、合并网络和纤维缓存，而非某个专门设计的工作点。因此，我们认为，将这些设计的关键特性与 SIGMA-like、SpArch-like 和 GAMMA-like 进行比较是合理的，因为我们的目标是确立灵活性的优势，并证明我们能够在不产生重大面积开销的情况下实现灵活性，而非获得某个专门设计的工作点。

对于这三种加速器，我们采用表 5 中所列的相同参数，只调整存储器控制器，使其按照相应的数据流以正确的顺序传送数据。我们还将 Flexagon 与英特尔数学核心函数库 [32] 进行比较；后者运行在一台 4 核 8 线程的英特尔酷睿 i5-7400 处理器 @ 3.00 GHz 上。每个核心均配备 128 KiB 的 L1 缓存、1 MiB 的 L2 缓存，以及一个共享的 6 MiB L3 缓存。我们未纳入图形处理器的结果，因为现有的图形处理器稀疏矩阵乘稀疏矩阵实现不原生支持稀疏权重+和激活值 [36, 33], , 因此其表现与 CPU MKL 相近，正如 [35, 34] 中所报告的那样。

为展示 Flexagon 的优势，我们的评估方法从以下三个不同角度进行考量：

端到端评估：为了真正证明 Flexagon 的性能优势，我们对完整的深度神经网络模型进行了端到端执行

[1]W我们计划在论文修订后将该框架开源，论文。

参数	值
乘法器数量	64
加法器数量	63
分发带宽	每周期 16 个元素
归约/合并带宽	每周期 16 个元素
总字长（值+坐标）	32 位
L1 访问延迟	1 个周期
L1 STA FIFO 大小	256 字节
L1 STR 缓存大小	1MiB
L1 STR 缓存行大小	128 字节
L1 STR 缓存相联度	16
L1 STR 缓存组数	16
PSRAM	256 KiB
DRAM 容量	16 GiB
DRAM 访问时间 / 带宽	100 纳秒 / 256 GB/s

表 5：Flexagon 的配置参数。

层	M、N、K	spA	spB	csA	csB	csC
SQ5	64、2916、16	68	11	1.2	162	728
SQ11	128、729、32	70	10	4.8	82	364
R4	256, 3136, 64	88	9	7.6	709	3136
R6	64, 2916, 576	89	53	16	3086	728
S-R3	64, 5329, 576	89	46	16	6422	1332
V0	128, 12100, 576	90	61	29	21357	12321
MB215	128, 8, 512	50	0	128	16	4
V7	512, 144, 4608	90	94	921	177	288
A2	384, 121, 1728	70	54	777	373	181

表 6：为评估选取的代表性 DNN 层。sp{A,B}=为矩阵 {A,B} 的稀疏性（以 % 计），cs{A,B,C}=为矩阵 {A,B,C} 的压缩大小（以 KiB 计）。

（见表2），在模拟的加速器上运行这些模型。这些模型包含在 MLPerf基准测试套件 [29] 中；为求全面，我们还选取了其他模型。可以看出，我们考察了种类非常多样的深度神经网络模型，其层数和规模各不相同。每个深度神经网络层执行时涉及的矩阵大小从 0.003 MiB 到 63.41 MiB 不等（见表 2中的平均压缩大小），因此我们的评估具有全面性，因为在许多情况下，矩阵无法完全放入芯片中（Flexagon 使用总计 1 MiB 的静态随机存取存储器来存储输入矩阵）。

逐层评估：由于解释结果需要深入到更细粒度的细节，我们从深度神经网络模型的执行过程中选取了 9 个代表性层。表6展示了这些层及其各自的特征。

RTL结果：我们实现了本研究所考虑加速器的主要构建模块（即 DN、MN、RN 和片上存储器），如表 7 所示。为了公平比较开销，四种架构的 DN 均采用相同的树形拓扑，MN 均采用相同的乘法器线性阵列，而 RN 则各不相同。对于类 SIGMA 架构，我们采用 FA N 网络 [5]作为 RN，以支持灵活大小的归约。对于类 Sparch 和类 GAMMA 架构，我们使用合并器 [35, 25] 来合并乘法后产生的部分和。最后，对于 Flexagon，我们采用第 3 节所述的统一 MRN。

在综合过程中，我们使用 MAERI BSV [1] 生成

	SIGMA-like	Sparch-like	GAMMA-like	Flexagon
DN	Tree	Tree	Tree	Tree
MN	线性	线性	线性	线性
RN	FAN	MergerS	MergerG	MRN

表 7：用于建模类 SIGMA、类 Sparch、类 GAMMA 和 Flexagon 加速器的主要构建模块。DN=配电网络，RN=归约网络，MN=乘法器网络。

64-MS 配电网络和乘法器网络。此外，我们还用寄存器传输级实现了一个 64 路合并器和我们的 MRN。我们分别使用 Synopsys Design Compiler 和 Cadence Innovus Implementation System 进行综合和布局布线，采用台积电 28nm GP 标准 LVT 库，频率为 800 MHz。为了获得存储结构的面积和功耗数据，我们使用 CACTI 7.0 [13]，针对相同的工艺节点和频率进行分析。

5. 结果

5.1. 端到端结果

图12 比较了运行 8 种深度神经网络模型时，中央处理器 MKL、三种当代固定数据流加速器（类 SIGMA、类 Sparch 和类 GAMMA）以及 Flexagon 的性能（相对于中央处理器 MKL 的结果所获得的加速比）。中央处理器 MKL 的总周期数列于表2中。

首先可以看出，没有一种固定数据流加速器能够在全部 8 种深度神经网络模型上都取得最高性能。具体而言，对于 *AlexNet*（A）、*VGG-16*（V）、*ResNet-50*（R）和 *SSD-ResNet*（S-R），类 Sparch 加速器的平均速度分别比类 SIGMA 和类 GAMMA 架构快 5.26× 倍和 1.49× 倍。相反，对于 *SqueezeNet*（SQ）、*SSD-MobileNet*（SM）、*DistilBERT*（DB）和 *MobileBERT*（MB），类 GAMMA 加速器的性能最佳（相较于类 SIGMA 和类 Sparch，平均性能分别提升 3.28× 倍和 2.41× 倍）。

第二个也是最值得注意的观察结果是，Flexagon 在所有情况下都优于其他三种固定数据流加速器，平均加速比分别达到 4.59×（相较于类 SIGMA）、1.71×（相较于类 Sparch）和 1.35×（相较于类 GAMMA）。这是因为它结合了灵活互连、显式解耦的存储结构和统一内存控制器，从而能够为每一层采用最高效的数据流。

最后，我们观察到，Flexagon 的性能显著优于 CPU MKL，因为该硬件专为执行稀疏矩阵乘稀疏矩阵运算而设计。总体而言，Flexagon 平均实现了 31× 的加速比（加速比最高可达 163×，性能提升幅度为 13× ）。

5.2. 逐层结果

要详细说明某些深度神经网络模型在特定数据流下观察到性能提升的原因，需要深入探究每一层的执行情况。为使研究切实可行

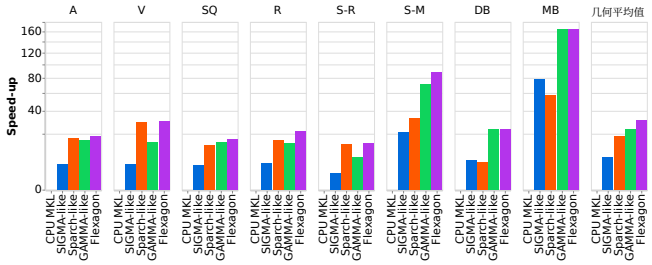


图 12：CPU MKL、类 SIGMA、类 Sparch、类 GAMMA 和 Flexagon 架构在 8 个深度神经网络模型上的性能比较（相对于类 SIGMA 的加速比）。

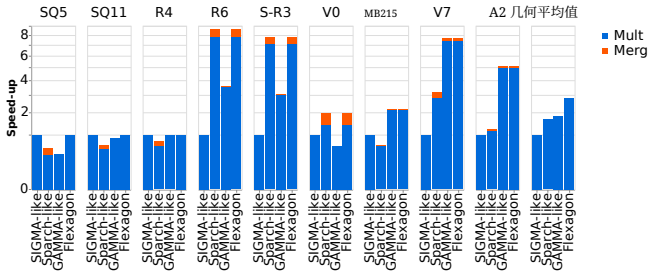


图 13：类 SIGMA、类 Sparch、类 GAMMA 和 Flexagon 架构在我们的 9 个深度神经网络层上的性能比较（相对于类 SIGMA 的加速比）。

（我们运行了上百个层），接下来，我们针对选出的九个具有代表性的深度神经网络层展开全面研究（表 6）。这些层根据其最能受益的数据流进行选择：表中前三层最受益于内积（*SQ5*、*SQ11* 和*R4*），接下来三层最受益于外积（*R6*、*S-R3* 和*V0*），最后三层最受益于Gust（*MB215*、*V7* 和*A2*）。

图 13 展示了使用我们模拟的加速器运行这些选定层时的性能比较（同样，加速比是相对于类 SIGMA 计算的）。如图所示，正如预期，在第一组适合内积的层中，类 SIGMA 架构相对于类 Sparch 和类 GAMMA 架构，平均加速比分别为 1.53×和 1.40×。接下来的三个适合外积的层（即 *R6*、*S-R3* 和*V0*）中，类 Sparch 架构相对于类 SIGMA 和类 GAMMA 架构，平均性能分别提升 5.07×和 2.66×。最后，在三个适合Gust的层中，类 GAMMA 架构性能最佳，其执行速度分别比类 SIGMA 和类 Sparch 架构快 4.37×和 3.19×。更值得注意的是，Flexagon 胜过了所有其他架构，始终达到最佳情况的性能。总体而言，根据每一层最合适的数据流恰当地配置 Flexagon 的控制逻辑后，我们的加速器相对于类 SIGMA、类 Sparch 和类 GAMMA 加速器的加速比分别可达 2.81×、1.69×和 1.55×。

图14、15 和16有助于我们理解这些结果。具体而言，图14展示了在我们片上

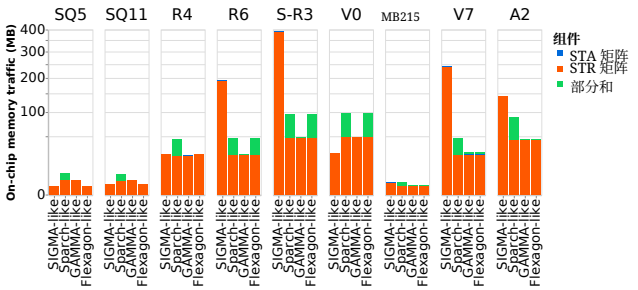


图 14：在我们的 9 个 DNN 层中，SIGMA、类 Sparch、类 GAMMA 和 Flexagon 架构的片上存储器层次结构中的存储器流量（MB）。

存储层次结构（即 STA FIFO 和 STR 缓存的读取，以及对 PSRAM 的读取/写入）与分发网络之间的片上存储器流量（以 MB 为单位）。该结果是在我们的九个 DNN 层上运行类 SIGMA、类 Sparch、类 GAMMA 和 Flexagon 架构后得出的。图15 绘制了这些层运行后的 STR 缓存未命中率，图16则展示了由此产生、在 STR 缓存与 DRAM 之间流动的片外流量（以 KB 为单位）。

我们从图14中得出的第一个观察结果是，从 STA 矩阵的存储器结构中取出的流量微乎其微（蓝色条形中占比极小）。这主要是因为驻留数据一经读取，便会在乘法器中保持驻留，直至执行结束，正如第3节所述。因此，无论配置何种数据流，这种存储器结构都不会对最终执行性能产生显著影响。相比之下，用于填充 STR 矩阵和 PSRAM 结构所需的流量会随层和数据流而大幅变化（分别对应橙色和绿色条形的部分），因此决定了该层的最终执行性能。

由于 内积 数据流无需合并部分和，因为部分和会在内部累加（请注意，类 SIGMA 架构发送到 PSRAM 的部分和数量始终为 0），因此这种数据流性能最佳。这一表现的一个例外是*V0*层。在此，类 SIGMA 架构中 STR 矩阵产生的流量低于类 Sparch 和类 GAMMA 架构。然而，该工作负载的运行时间却更长。原因是矩阵 B 的尺寸很大（21.3 MiB），导致它必须重新加载多次，L1 未命中率达到 3.13%（见图 15），显著高于类 Sparch 和类 GAMMA 架构的 L1 未命中率（即 0.36% 和 2.30%），从而增加了片外存储器流量（见图16）。这一更高的流量使得类 SIGMA 架构的乘法阶段耗时，比类 Sparch 架构的乘法和合并阶段总耗时还长。当交点数量较少时，类 SIGMA 架构

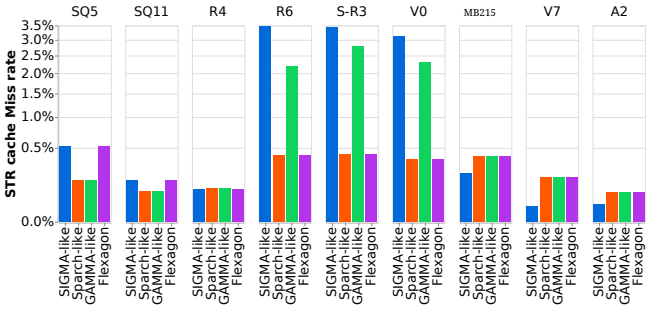


图 15：SIGMA 类、Sparch 类、GAMMA 类和 Flexagon 架构在 9 个深度神经网络层上的 STR 缓存未命中率。

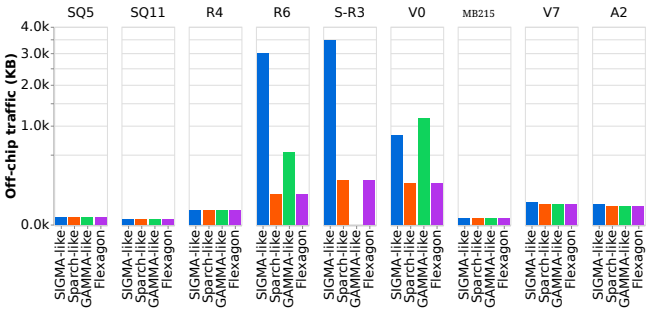


图 16：SIGMA 类、Sparch 类、GAMMA 类和 Flexagon 架构在 9 个深度神经网络层上的片外数据流量。

由于该架构会访问更多数据元素，因此运行时会产生更多周期开销。六个无法从类 SIGMA 架构中受益的层也体现了这一点（即 *R6*、*S-R3*、*V0*、*MB215*、*V7* 和 *A2*），其片上流量平均分别比类 Sparch 和类 GAMMA 架构高 5.68× 和 2.27×。

另一方面，在这六层中，决定它们性能差异的主要因素是矩阵 B 的大小。第二组层（即 *R6*、*S-R3*和 *V0*）受益于 Sparch 类架构，因为其矩阵 B 较大（见表 6）。这意味着 GAMMA 类架构无法将 B 的各行完整地装入 STR 矩阵的存储器结构中，从而导致更高的 L1 未命中率。观察这三层执行过程中测得的平均 L1 未命中率（见图 15）：Sparch 类架构为 0.39%，GAMMA 类架构为 2.43%。这使 GAMMA 的流量增加了 6.25×，导致性能下降。

在最后一组层（即 *MB215*、*V7* 和 *A2*）中，矩阵 B 的尺寸小得多（如表6所示，最大为 373KB），因此，类 Sparch 和类 GAMMA 架构具有相同的 L1 未命中率和片外数据流量。在这种情况下，类 GAMMA 架构效率更高，因为它能够同时执行合并阶段和合并阶段——请注意图13 中类 GAMMA 情况下的橙色柱并不显著，因为合并阶段与乘法阶段（即蓝色柱）并行计算。

组件	SIGMA-like	Sparch-like	GAMMA-like	Flexagon
面积结果				
分发网络 (mm^2)	0.04	0.04	0.04	0.04
乘法器网络 (mm^2)	0.07	0.07	0.07	0.07
归约/合并网络 (mm^2)	0.17	0.07	0.07	0.21
缓存 (mm^2)	3.93	3.93	3.93	3.93
PSRAM (mm^2)	-	1.03	0.51	1.03
总计 (mm^2)	4.21	5.14	4.62	5.28
功耗结果				
分发网络 (mW)	2.18	2.18	2.18	2.18
乘法器网络 (mW)	3.29	3.29	3.29	3.29
归约/合并网络 (mW)	248	64.48	64.48	312
缓存 (mW)	2142	2142	2142	2142
PSRAM (mW)	-	538	269	538
总计（mW）	2396	2750	2481	2998

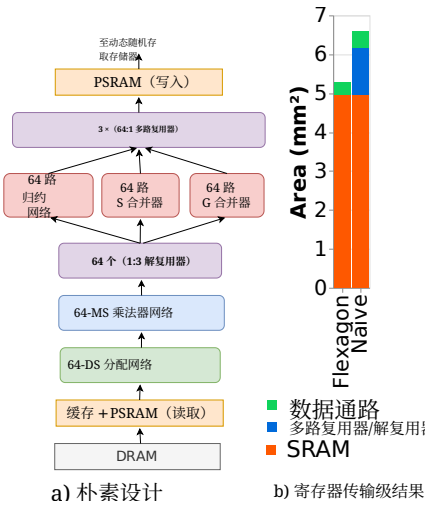
表8：类 SIGMA、类 Sparch、类 GAMMA 和 Flexagon加速器的后布局面积与功耗。

5.3. 寄存器传输级结果

表8 展示了 64-MS 的类 SIGMA、类 Sparch、类 GAMMA 和 Flexagon 加速器的总面积（mm^2）和功耗（mW）细分结果。对于每种情况，我们展示了主要架构组件的结果：分配网络（DN）、乘法器网络（MN）、归约/合并网络（RN）、流式矩阵的缓存结构（缓存）以及 PSRAM。

就面积而言，与类 SIGMA、类 Sparch 和类 GAMMA 加速器的面积相比，Flexagon 的开销分别增加了 25%、3% 和 14%。可以看到，四种加速器的面积主要由存储结构决定。具体而言，流式矩阵的缓存分别占类 SIGMA、类 Sparch、类 GAMMA 和 Flexagon 架构总面积的 93%、76%、85% 和 74%。此外，PSRAM 的面积分别占类 Sparch、类 GAMMA 和 Flexagon 加速器面积的 20%、11% 和 19%。由于类 SIGMA 架构采用内积 数据流，因此该加速器不需要这种结构，这也解释了它的面积为何最小。此外，类 GAMMA 加速器的 PSRAM 面积仅为类 Sparch 和 Flexagon 加速器的一半，因为它需要存储的部分和更少，这也解释了面积为何有所降低。显然，Flexagon 需要支持最坏情况的外积 数据流，因此 PSRAM 开销最高。最后请注意，我们的 MRN 面积比 FAN 和合并器分别大 28% 和 128%，但这并未造成很高的整体开销，因为 MRN 仅占 Flexagon 总面积的 4%。

图 17 证明了将 RN 和合并器统一为单个网络（MRN）所带来的面积优势。为此，我们绘制了一个与 Flexagon 类似的 64-MS 朴素加速器设计，但针对每种数据流分别使用独立网络（见图17a）。我们在此使用朴素一词，是为了强调该设计只是将归约网络复制了 3 次（每种数据流各一次）。如图所示，归约网络和合并器网络共用相同的乘法器网络和分布网络，以及相同的 SRAM 容量。该设计



a) 朴素设计　　b) 寄存器传输级结果

图17：a）非统一朴素设计的高层概览。b）Flexagon 与朴素设计的面积比较。

需要额外的连线、多路复用器和解复用器来连接各个部分。在底部，MN 连接到三个不同的网络，因此需要 64 个（1:3）解复用器。在顶部，合并与归约网络中的每个节点都必须连接到存储器，这需要 3 个成本高昂的（64:1）多路复用器及相应连线。图 17b 展示了这种朴素设计的低效之处。可以看到，三个独立网络（即 RN 和合并器）仅带来 2% 的面积开销，因为设计的面积主要由 SRAM 占据（例如，Flexagon 中 SRAM 占 74%）。朴素设计造成的显著面积代价源于额外的多路复用器、解复用器及相应连线，使其相较 Flexagon 的面积开销达到 25%。请注意，在更大的配置中（即乘法器数量更多时），这一面积开销还会进一步增加。

在功耗方面，我们观察到了相同的趋势。我们发现，Flexagon加速器的功耗分别比类SIGMA、类 Sparch和类GAMMA加速器高出25%、9%和21%。相较于前述面积结果，Flexagon略高的开销主要来自合并器/RN，因为该模块占总功耗的比例更大（在类SIGMA、类 Sparch、类GAMMA和Flexagon加速器中，分别观察到10%、2.34%、2.60%和10.41%的占比）。此外，MRN的功耗比FAN RN和合并器分别高出25%和284%，这也解释了上述结果。尽管引入了额外开销，但在图18中我们展示了Flexagon仍具有更高的性能/面积效率。具体而言，我们同时考虑了每种设计所达到的加速比和面积需求。面积需求以类SIGMA方案为基准进行归一化，加速比的计算也以该方案为参考。请注意，类似*MobileBert*（*MB*）和*DistilBert*（*DB*）的自然语言处理模型在类 GAMMA加速器上实现了更高的效率。然而，正如前文所述，这是因为这些模型的大多数层（DistilBert（*DB*）中有84%，MobileBert（*MB*）中有100%）在使用 Gustavson

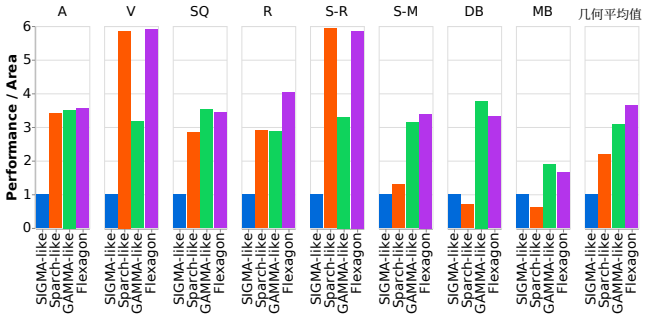


图18：在我们的 8 个深度神经网络模型上运行类 SIGMA、类 Sparch、类 GAMMA 和 Flexagon 架构后得到的性能/面积。

数据流时表现更好，因此Flexagon加速器引入的面积开销并无必要。因此，我们可以清楚地看到，总体而言，Flexagon在性能和面积消耗之间实现了最佳折中（具有更高的加速比/面积值）。相比之下，我们发现，在执行8个深度神经网络模型期间，与类 GAMMA、类 Sparch和类SIGMA加速器相比，我们的加速器平均分别取得了18%、67%和265%的更高性能/面积效率。这使Flexagon成为运行异构稀疏深度神经网络工作负载的最佳候选方案。

6. 相关工作

稀疏深度神经网络加速器：稀疏矩阵乘法一直是人工智能和高性能计算工作负载加速的主要目标。针对稀疏矩阵乘法、稀疏矩阵乘法以及稀疏卷积，已有多种稀疏深度神经网络加速器被提出[10, 25, 12, 31, 2, 20, 8, 5, 16, 26]。这些加速器支持稀疏执行，可将一个或两个操作数压缩为压缩稀疏行、压缩稀疏列、位图、压缩稀疏纤维格式等格式。这减少了内存占用和乘法次数。如表 1 所示，先前的稀疏加速器采用了内积、外积 和Gust（按行乘积）数据流中的一种。我们表明，支持多种数据流的灵活性有助于提升性能和单位面积性能。

灵活加速器框架： 灵活性方面的先前工作包括硬件模块以及面向粗粒度可重构阵列的设计空间探索工具。MINT [28] 是一种支持多种稀疏格式的格式转换组件。Garg 等人 [7]，、coSPARSE [6] 和 SparseAdapt [24] 提出了用于在粗粒度可重构阵列上高效执行稀疏计算的框架。然而，据我们所知，这是首项提出用于稀疏深度神经网络的加速器、并充分利用三种数据流的工作。

7. 结论

本研究提出 Flexagon，这是首个在同质硬件底层平台上支持内积、外积 和Gust 数据流的稀疏矩阵乘稀疏矩阵加速器设计。Flexagon 的核心是一种新颖的树形网络（MRN），既支持点积归约，也支持部分和合并；此外，它还采用一种特殊的片上 L1 存储器组织，专门针对输入和输出压缩矩阵不同的访问特性进行设计。

通过采用最适合各深度神经网络层特性的 dataflow，我们证明，与最先进的固定数据流稀疏加速器相比，Flexagon 显著提升了性能/面积效率。

参考文献

[1] MAERI代码 v1。https://github.com/hyoukjun/MAERI。 [2] Yu-Hsin Chen、Tien-Ju Yang、Joel Emer 和 Vivienne Sze。Eyeriss v2：面向移动设备中新兴深度神经网络的灵活加速器。*IEEE*新兴与精选电路和系统专题期刊，9(2):292–308，2019年6月。 [3] Jacob Devlin、Ming-Wei Chang、Kenton Lee 和 Kristina Toutanova。BERT：用于语言理解的深度双向变换器预训练。*arXiv*预印本 *arXiv: 1810.04805v2 (2019)*，2019年5月。 [4] A. Parashar 等。Timeloop：一种系统化的深度神经网络加速器评估方法。国际系统与软件性能分析研讨会，2019年3月。 [5] Eric Qin 等。SIGMA：一种面向深度神经网络训练、具备灵活互连的稀疏非规则通用矩阵乘法加速器。国际高性能计算机体系结构研讨会，2020年3月。 [6] Siying Feng、Jia wen Sun、Subhankar Pal、Xin He、Kuba Kaszyk、Dong-Hyeon Park、Magnus Morton、Trevor Mudge、Murray Cole、Michael O'Boyle、Chaitali Chakrabarti 和 Ronald Dreslinski。Cosparse：用于图分析的软硬件可重构稀疏矩阵向量乘法框架。载于*2021*年第*58*届*ACM/IEEE*设计自动化会议（*DAC*），第949–954页，2021年。 [7] Raveesh Garg、Eric Qin、Francisco Muñoz-Martínez、Robert Guirado、Akshay Jain、Sergi Abadal、José L Abellán、Manuel E Acacio、Eduard Alarcón、Sivasankaran Rajamanickam 和 Tushar Krishna。理解空间加速器上稀疏/稠密多阶段图神经网络数据流的设计空间。载于*2022*年*IEEE*国际并行与分布式处理研讨会（*IPDPS*），2022年。 [8] Ashish Gondimalla、Noah Chesnut、Mithuna Thottethodi 和 TN Vijaykumar。Sparten：用于卷积神经网络的稀疏张量加速器。载于第*52*届*IEEE/ACM*国际微体系结构研讨会论文集，第151–165页，2019年。 [9] Fred G. Gustavson。稀疏矩阵的两种快速算法：乘法与置换转置。*ACM*数学软件汇刊，4(3):250–269，1978年9月。 [10] S. Han、X. Liu、H. Mao、J. Pu、A. Pedram、M. A. Horowitz 和 W. J. Dally。EIE：面向压缩深度神经网络的高效推理引擎。载于 *2016* 年第*43*届*ACM/IEEE*国际计算机体系结构年会（*ISCA*），第243–254页，2016年6月。 [11] Song Han、Huizi Mao 和 William J. Dally。深度压缩：通过剪枝、训练后量化和霍夫曼编码压缩深度神经网络。*arXiv*预印本 *arXiv: 1510.00149v5 (2016)*，2016年2月。 [12] Kartik Hegde、Hadi Asghari-Moghaddam、Michael Pellauer、Neal Crago、Aamer Jaleel、Edgar Solomonik、Joel Emer 和 Christopher W Fletcher。ExTensor：一种稀疏张量代数加速器。载于第*52*届*IEEE/ACM*国际微体系结构研讨会论文集，第319–333页，2019年。 [13] 惠普实验室。CACTI 7.0：用于建模缓存/存储器、三维堆叠和片外输入输出的工具。https://github.com/HewlettPackard/cacti。 [14] Janssen、Curtis L. 等。一种用于大规模并行计算机体系结构的模拟器。*IJDST* 第*1*卷第*2*期，第57–73页，2010年。 [15] Norman P. Jouppi 等。张量处理单元的数据中心内性能分析。载于第*44*届国际计算机体系结构研讨会，2017年。 [16] Konstantinos Kanellopoulos、Nandita Vijaykumar、Christina Giannoula、Roknoddin Azizi、Skanda Koppula、Nika Mansouri Ghiasi、Taha Shahroodi、Juan Gomez Luna 和 Onur Mutlu。Smash：协同设计软件压缩与硬件加速索引，以实现高效的稀疏矩阵运算。载于第*52*届*IEEE/ACM*国际微体系结构研讨会论文集，第600–614页，2019年。 [17] Fredrik Kjolstad、Stephen Chou、David Lugato、Shoaib Kamil 和 Saman Amarasinghe。TACO：用于生成张量代数内核的工具。第*32*届*IEEE/ACM*国际自动化软件工程会议论文集，第943–948页，2017年10月。 [18] Hyoukjun Kwon 等。理解深度神经网络数据流的复用、性能与硬件成本：一种以数据为中心的方法。国际微体系结构研讨会（*MICRO*），2019年10月。

[19] Hyoukjun Kwon、Ananda Samajdar 和 Tushar Krishna。MAERI：通过可重构互连实现深度神经网络加速器上的灵活数据流映射。编程语言与操作系统架构支持国际会议，2018年3月。 [20] Ching-En Lee、Yakun Sophia Shao、Jie-Fang Zhang、Angshuman Parashar、Joel Emer、Stephen W Keckler 和 Zhengya Zhang。Stitch-x：一种利用深度神经网络中非结构化稀疏性的加速器架构。载于*SysML*会议，第120卷，2018年。 [21] Peter Mattson、Christine Cheng、Cody Coleman、Greg Diamos、Paulius Micikevicius、David Patterson、Hanlin Tang、Gu-Yeon Wei、Peter Bailis、Victor Bittorf 等。MLPerf训练基准。*arXiv*预印本 *arXiv:1910.01500*，2019年。 [22] F. Muñoz-Martínez 等。STONNE：支持深度神经网络推理加速器的周期级微体系结构模拟。载于*IISWC*，2021年。 [23] Maxim Naumov、Dheevatsa Mudigere、Hao-Jun Michael Shi、Jianyu Huang、Narayanan Sundaraman、Jongsoo Park、Xiaodong Wang、Udit Gupta、Carole-Jean Wu、Alisson G. Azzolini、Dmytro Dzhulgakov、Andrey Mallevich、Ilia Cherniavskii、Yinghai Lu、Raghuraman Krishnamoorthi、Ansha Yu、Volodymyr Kondratenko、Stephanie Pereira、Xianjie Chen、Wenlin Chen、Vijay Rao、Bill Jia、Liang Xiong 和 Misha Smelyanskiy。用于个性化和推荐系统的深度学习推荐模型。*CoRR*，abs/1906 00091，2019年。。[24] Subhankar Pal、Aporva Amarnath、Siying Feng、Michael O'Boyle、Ronald Dreslinski 和 Christophe Dubach。SparseAdapt：用于可重构加速器上稀疏线性代数的运行时控制。载于*MICRO-54*：第*54*届*IEEE/ACM*国际微体系结构研讨会，MICRO '21，第1005–1021页，美国纽约，2021年。计算机协会。 [25] Subhankar Pal、Jonathan Beaumont、Dong-Hyeon Park、Aporva Amarnath、Siying Feng、Chaitali Chakrabarti、Hun-Seok Kim、David Blaauw、Trevor Mudge 和 Ronald Dreslinski。OuterSPACE：一种基于外积的稀疏矩阵乘法加速器。载于*ISCA*，2018年。[26] Angshuman Parashar、Minsoo Rhu、Anurag Mukkara、Antonio Puglielli、Rangharajan Venkatesan、Brucek Khailany、Joel Emer、Stephen W. Keckler 和 William J. Dally。SCNN：一种用于压缩稀疏卷积神经网络的加速器。国际计算机体系结构研讨会（*ISCA*），第27–40页，2017年6月。 [27] Eric Qin、Geonhwa Jeong、William Won、Sheng-Chun Kao、Hyoukjun Kwon、Sudarshan Srinivasan、Dipankar Das、Gordon E Moon、Sivasankaran Rajamanickam 和 Tushar Krishna。扩展稀疏张量加速器以支持多种压缩格式。载于*2021*年*IEEE*国际并行与分布式处理研讨会（*IPDPS*），第1014–1024页。IEEE，2021年。[28] Eric Qin、Geonhwa Jeong、William Won、Sheng-Chun Kao、Hyoukjun Kwon、Sudarshan Srinivasan、Dipankar Das、Gordon E. Moon、Sivasankaran Rajamanickam 和 Tushar Krishna。扩展稀疏张量加速器以支持多种压缩格式。载于*2021*年*IEEE*国际并行与分布式处理研讨会（*IPDPS*），第1014–1024页，2021年。[29] V. J. Reddi 等。MLPerf推理基准。第*47*届国际计算机体系结构研讨会（*ISCA*），2020年5月。[30] Vijay Janapa Reddi、Christine Cheng、David Kanter、Peter Mattson、Guenther Schmuelling、Carole-Jean Wu、Brian Anderson、Maximilien Breughe、Mark Charlebois、William Chou 等。MLPerf推理基准。载于*2020*年*ACM/IEEE*第*47*届年度国际计算机体系结构研讨会（*ISCA*），第446–459页。IEEE，2020年。[31] Nitish Srivastava、Hanchen Jin、Jie Liu、David Albonesi 和 Zhiru Zhang。MatRaptor：一种基于按行乘积的稀疏-稀疏矩阵乘法加速器。载于*2020*年第*53*届*IEEE/ACM*国际微体系结构研讨会（*MICRO*），第766–780页。IEEE，2020年。[32] Endong Wang、Qing Zhang、Bo Shen、Guangyong Zhang、Xiaowei Lu、Qing Wu 和 Yajuan Wang。英特尔数学核心函数库。英特尔至强融核上的高性能计算，2014年6月。 [33] Ziheng Wang。SparseRT：加速图形处理器上的非结构化稀疏性以实现深度学习推理。*arXiv*预印本 *arXiv: 2008.11849v1 (2020)*，2020年8月。 [34] Guowei Zhang、Nithya Attaluri、Joel S Emer 和 Daniel Sanchez。GAMMA：利用Gustavson算法加速稀疏矩阵乘法。载于第*26*届*ACM*编程语言与操作系统架构支持国际会议论文集，第687–701页，2021年。[35] Zhekai Zhang、Hanrui Wang、Song Han 和 William J. Dally。Sparch：高效的稀疏矩阵乘法架构。国际高性能计算机体系结构研讨会（*HPCA*），第261–274页

, 2020年2月。

[36] Maohua Zhu 和 Yuan Xie. 通过延迟感知优化驾驭图形处理器上的非结构化稀疏性。 *2020*年 第*57*届 *ACM/IEEE* 设计自动化会议（*DAC*）， 2010年10月。