Bit-balance：利用位级稀疏性加速神经网络的模型-硬件协同设计

Wenhao Su n、Zhiwei Zou、Deng Liu、Wendi Sun、Song Chen，*IEEE*会员，以及 Yi Kang

摘要—位串行架构能够处理具有不同权重精度的神经网络（NN），与位并行架构相比，资源效率更高。此外，由于神经网络具有容错性，权重中包含大量零位，这表明可以进一步利用神经网络的位稀疏性来提升性能。然而，每个权重中零位比例不规则，导致处理单元（PE）阵列的工作负载不均衡，从而降低性能或增加稀疏处理开销。因此，本文提出一种位稀疏量化方法，将每个权重的位稀疏率限制在某一阈值以内，以平衡工作负载，同时仅造成很小的准确率损失。随后，我们协同设计了一种名为 Bit-balance 的稀疏位串行架构，以提升整体性能，并支持权重位稀疏度和自适应位宽计算。该设计采用 65纳米工艺，在 1 GHz 下实现，AlexNet、VGG-16、ResNet-50 和 GoogleNet 的性能分别达到326、30、56和218帧/秒。与稀疏位串行加速器 Bitlet 相比，Bit-balance 的能效（帧/J）提升 1.8×~2.7×，资源效率（帧/mm²）提升2.1×~3.7×。

索引词—硬件加速器、位稀疏性、量化、神经网络。

arXiv:2302.00201v1 [cs.AR] 1 Feb 2023

1 引言

如今，神经网络已广泛应用于众多领域，例如图像识别 [1], [2], [3], [4], [5]语音识别 [6],目标检测 [5], 和计算机视觉 [7]。从结构上看，神经网络主要由卷积（CONV）层和全连接（FC）层组成；前者执行卷积运算，后者执行矩阵-向量乘法（GEMV），二者都包含大量乘加运算（MAC）。随着对精度和复杂度的要求不断提高，网络计算的工作负载也持续增加。因此，研究人员一直致力于开发神经网络硬件加速器，以跟上软件的发展步伐。

尽管神经网络带来了大量计算，但其中也存在单调重复的运算；因此，许多硬件架构师试图通过提高计算并行度来缩短运行时间。加速器可分为四类：位并行、稀疏位并行、位串行和稀疏位串行。DaNN [8],Eyeriss [9], TPU [10], 天机 [11], 等是典型的位并行加速器，主要通过配备大量定点计算单元来提高并行度，从而实现加速。此外，由于神经网络具有容错性，输入特征图（IFM）和权重中包含大量零值，因此可以通过跳过零值计算来加速神经网络。由此涌现出许多稀疏位并行加速器，例如 EIE [12],寒武纪-X[13], Cnvlutin [14],SCNN [15], 等，进一步提升了性能和能效。然而，这些工作主要是针对定点计算而设计的，未考虑不同神经网络的权重/输入特征图精度各不相同，导致资源利用率和能效低下。

为适应不同神经网络的不同精度，研究人员尝试设计基于位串行计算的加速器。Stripe [16] 提出了一种依赖位串行计算单元的硬件加速器，并在每一层将输入特征图量化到所需精度，p，与 16 位定点基准相比可实现 $16/p\times$ 加速比。Loom [17] 通过减少权重和输入特征图的位数来实现位串行加速。Bit-Fusion[18] 提出了一种位宽灵活加速器，以匹配每个独立乘加操作数的位宽，从而最大限度地提高不同位宽运算的资源效率。UNPN [19] 设计了一种基于查找表（LUT）的位串行处理单元，与传统定点乘加单元相比，能耗降低了 23% 至 54%。这些加速器利用位串行计算来提升性能和能效。然而，它们并未利用位稀疏性，这表明性能仍有提升空间。

由于神经网络具有位稀疏性，一些研究人员设计了稀疏位串行加速器，以进一步提升性能。Pragmatic [20] 跳过了参与计算的输入特征图元素在相同零位位置上的计算。然而，每个输入特征图位序列中的零位分布并不规则，因此只能截断一小部分零位。Laconic [21] 将乘加运算拆解为输入特征图和权重的位串行计算，并通过利用输入特征图和权重的稀疏性来加速。不过，输入特征图和权重中的零位分布可能并不规则，导致处理单元阵列中的工作负载不均衡，从而降低性能。Bit-Tactical [22] 采用权重跳过模块和输入特征图位选择模块，通过前端调度优化不规则稀疏权重的布局，

• *Wenhao Sun*、*Zhiwei Zou*、*Deng Liu*、*Wendi Sun*、*Song Chen* 和 *Yi Kang* 均隶属于中国科学技术大学微电子学院，地址为中国安徽省合肥市 *30332*。电子邮箱：*wh1997@mail.ustc.edu.cn*；*songch@ustc.edu.cn*

从而进一步避免对零权重元素和输入特征图位进行计算。然而，处理单元之间输入特征图的独立供给使其数据缓存资源远远超过计算资源的面积，导致资源利用率低。Bitlet [23]提出位交织理念，以最大程度地利用位级稀疏性，并通过减少参与计算的权重数量来实现加速。然而，与位交织对应的逻辑占整个处理单元模块面积的40%以上。此外，高精度计算单元在低精度操作期间可能处于闲置状态，导致资源效率低下。尽管这些架构利用位稀疏性实现加速，但仍受困于工作负载不均衡，或稀疏位串行处理实现开销过大的问题。

本文旨在平衡处理单元阵列中的稀疏位串行工作负载，同时降低稀疏性处理的开销。我们提出了一种面向神经网络的稀疏位串行加速器的模型-硬件协同设计，以提升系统性能和能效。我们的主要贡献如下：

1). 我们提出了一种基于脉动阵列的加速器，称为Bit-balance，支持权重的稀疏位串行处理和自适应位宽计算；通过直接处理特定编码格式的稀疏权重，无需额外的预处理模块即可实现高性能。

2). 我们协同设计了一种位稀疏量化方法，在精度损失较小的情况下，将每个权重的稀疏率保持在特定值，从而平衡处理单元负载，并显著提升性能。

与位串行加速器相比，Stripe [16]，Bit-balance 利用权重位稀疏性实现加速，取得了 $4\times$~$7.1\times$ 加速比和 $3\times$~$5.6\times$ 能效。此外，与稀疏位串行加速器相比，Laconic [21]，平衡了处理单元阵列中的工作负载，取得了 $2.2\times$~$4.3\times$ 加速比和 $2.7\times$~$5.4\times$ 能效。此外，与 Bitlet [23]，相比，我们显著降低了稀疏处理的开销，实现了 $1.8\times$~$2.7\times$ 能效和 $2.1\times$~$3.8\times$ 资源效率。

2 预备知识

由于神经网络可应用于各种领域，权重的精度也各不相同。此外，由于具有较强的容错性，神经网络的权重中包含大量零位。因此，架构在.....方面有很大的潜力

基于位串行计算获得更多收益。尽管位宽量化可以直接减少位串行计算的周期，但为了维持准确率，可削减的位宽有限。利用权重的位稀疏性可以进一步加速位串行计算，但处理单元阵列会面临工作负载不均衡的问题。处理非零位数（NNZB）较少权重的处理单元必须等待处理非零位数较多权重的处理单元，从而降低性能和处理单元利用率。因此，关键挑战是平衡处理单元阵列中的稀疏位串行计算工作负载。

2.1 神经网络的计算过程

神经网络的计算过程如图1所示。卷积层接收大小为 $H_i \times W_i$的输入特征图的 C_i 个输入通道（IC）。通过在一个输入特征图上滑动大小为 $H_k \times W_k$ 的卷积核窗口，并累加 C_i 个输入通道的临时结果，生成大小为 $H_o \times W_o$的 C_o 个输出通道（OC）对应的每个输出特征图。最后，通过对共享输入特征图的 C_o 组卷积核进行卷积，计算得到 C_o 个输出特征图。对于全连接层，输入特征图和输出特征图的大小均为 1×1。将输入特征图设为矩阵 $I[C_i,H_i,W_i]$，权重设为 $W[C_o,C_i,H_k,W_k]$，输出特征图设为 $O[C_o,H_o,W_o]$，则该过程可表示为公式1。

$$O[o,x,y]=\sum_{i=0}^{C_i}\sum_{a=0}^{H_k}\sum_{b=0}^{H_k} I[i,x+a,y+b]\times W[o,i,a,b] \quad (0\le o<C_o, 0\le x<H_o, 0\le y<W_o) \tag{1}$$

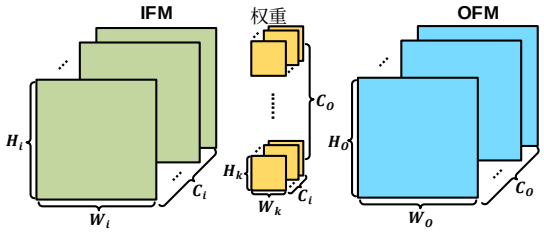

图 1：卷积过程。

参见公式 1，神经网络处理的基本运算是乘加运算。假设权重的位宽为 N，则输入特征图 I_0 与权重 W_0 的乘加运算可分解为移位与加法运算，如公式 2 所示，这种方式能够灵活适应不同的位宽。

$$I_0 \times W_0 = \sum_{i=0}^{N} I_0 \times W_0[i] \times 2^i \tag{2}$$

2.2 稀疏位串行计算的挑战

计算

要加速位串行处理，最直接的方法是将权重量化至更低的位宽。如果将权重从 N 位量化为 N_{pb} 位，就能实现 $N/N_{pb}\times$ 的加速比。图 2 展示了采用量化权重的位串行计算示例，其性能提升达到 $1.3\times$。然而，假设 N 位权重的取值为 2^N，当位宽调整为 N_{pb} 时，该值可能降至 $2^{N_{pb}}$。为保持准确率，位宽的缩减幅度可能受到限制，从而制约性能提升。

为进一步提升性能，可以利用权重的位稀疏性，通过跳过零位计算来加速。假设每个权重的非零位数为 N_{nzb}，最大值为 N_{nzb_max}，权重经过压缩后加载到处理单元阵列中，计算周期为 N_{nzb_max}。图 3(a) 展示了利用稀疏性进行位串行计算的示例，性能达到 $1.6\times$。然而，由于权重的稀疏率呈随机分布，处理单元阵列中的工作负载并不均衡。非零位数较少的处理单元必须等待非零位数较多的处理单元完成。因此，PE_1 将空闲 2

个周期，表明处理单元利用率低下。此外，一旦 N_{nzb_max} 等于原始权重位宽，性能便无法提升。

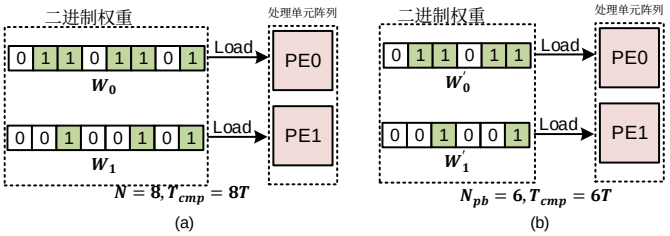

图 2：在处理单元阵列中使用原始权重和量化权重进行位串行计算。(a) 原始权重。(b) 量化权重。

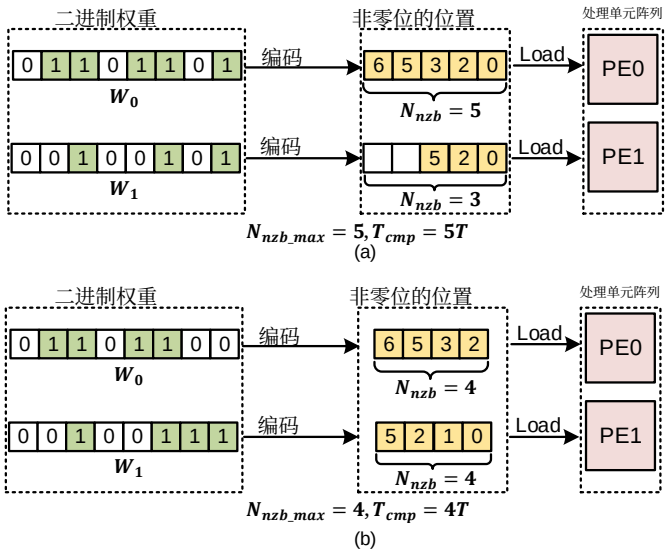

图 3：处理单元阵列中非零权重位工作负载不均衡与均衡时的稀疏位串行计算。(a) 工作负载不均衡。(b) 工作负载均衡。

尽管已有一些加速位串行计算的工作，例如 Stripe [16], Laconic [21]和 Bitlet [23], ，但它们都因工作负载不均衡而导致性能下降或资源效率低下。因此，在本工作中，我们旨在均衡处理单元阵列中稀疏权重位的工作负载。

3 方法

为解决处理单元阵列中的工作负载不均衡问题，我们基于位稀疏性对权重进行量化，并将每个权重中的非零位数限制在某个特定值以内，从而显著提升处理单元利用率和性能。为适配位串行计算，我们以编码格式存储权重，以便直接用于移位加法运算。

3.1 位稀疏量化

由于神经网络具有较强的容错性，权重中允许存在冗余位。然而，直接将权重量化为较低位宽会显著降低权重值的数值范围，而利用权重随机分布的位稀疏性则会导致工作负载不均衡。因此，为了保持权重值的数值范围并平衡处理单元阵列中的计算负载，我们提出了一种位稀疏量化方法，将若干权重位设为零，并使每个权重中的最大非零位数 N_{nzb_max} 不超过某个特定值。假设原始权重的位宽为 N，其数值范围可计算为 $\sum_{i=0}^{N_{nzb_max}} \binom{N}{i}$。与直接将权重量化为 N_{pb} 位相比，我们的方法能提供更大的权重值数值范围，如表1所示。位稀疏量化中 $^-$ 位的 $N_{nzb\ max} = 3$ 情形，可以与位宽量化中的 $N_{pb} = 9$ 相媲美。

与利用权重中随机分布的位稀疏性相比，我们的量化方法具有更高的性能，这是因为稀疏权重位在处理单元阵列中的工作负载更加均衡。位稀疏量化计算流程的一个示例如下如图2(b)所示， W_0 和 W_1 中的非零位数均为四，从而消除了空闲的处理单元。因此，与不均衡工作负载情况相比，我们获得了 $4T_w$ 的总计算时间，并实现了 $1.25\times$ 的加速比。

位稀疏量化的具体流程如图4所示。首先，我们为权重设置一个初始最大非零位数 NNZB， N_{nzb_max}。然后，对于 NNZB 大于 N_{nzb_max} 的权重，我们将低有效非零位设为零，并将非零位总数保持为 $N_{nzb\ max}$。接下来，重新训练此前量化的权重，并验证准确率是否超出边界。如果没有，我们继续降低 N_{nzb_max} 并训练权重；否则，我们保存最终的量化权重并记录 N_{nzb_max}。图5展示了两个 8 位权重的量化示例，W_0 和 W_1。我们将每个权重的 NNZB 设为小于4。尽管零位的分布不规则，但每个权重的工作负载是均衡的，因此实现了$8/4 = 2\times$加速比。

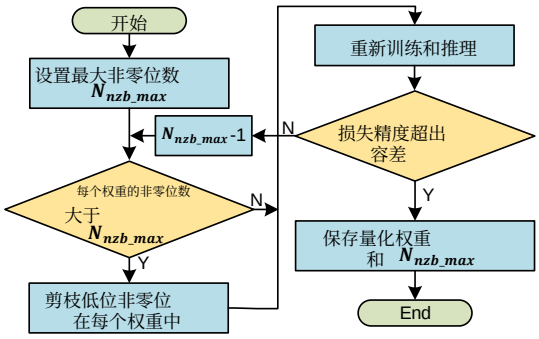

图 4：位稀疏量化的流程。

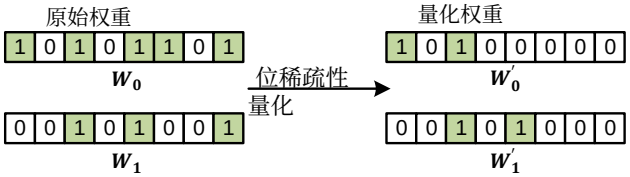

图 5：位稀疏量化示例。

表1：权重值的数值范围比较

位宽量化	N_{pb}	13	12	11	10	9	8	7	6	5	4	3
	$2^{N_{pb}}$	8192	4096	2048	1024	512	256	128	64	32	16	8
位稀疏性量化	(N_{nzb_max},N)	(13,16)	(12,16)	(11,16)	(10,16)	(9,16)	(8,16)	(7,16)	(6,16)	(5,16)	(4,16)	(3,16)
	$\sum_{i=0}^{N_{nzb_max}}\binom{N}{i}$	65339	64839	63019	58651	50643	39203	26333	14893	6885	2517	697

3.2 稀疏位串行处理

为降低稀疏性处理的开销，权重采用长度、符号位、位位置和位图进行编码，如图6所示。长度 N_{nzb_max}表示所有权重的最大非零位数 NNZB，在当前层中只需为所有权重存储一次。符号位 W_s表示一个权重为正还是为负。位位置 W_p 决定输入特征图 IFM 应移位多少位。位图 W_b表示该位位置是否有效，因为某些权重的 NNZB 小于$N_{nzb\ max}$。

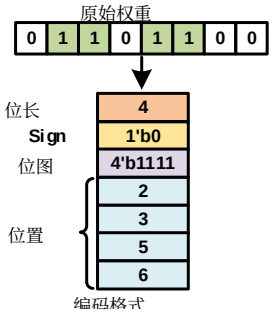

图 6：权重编码格式。

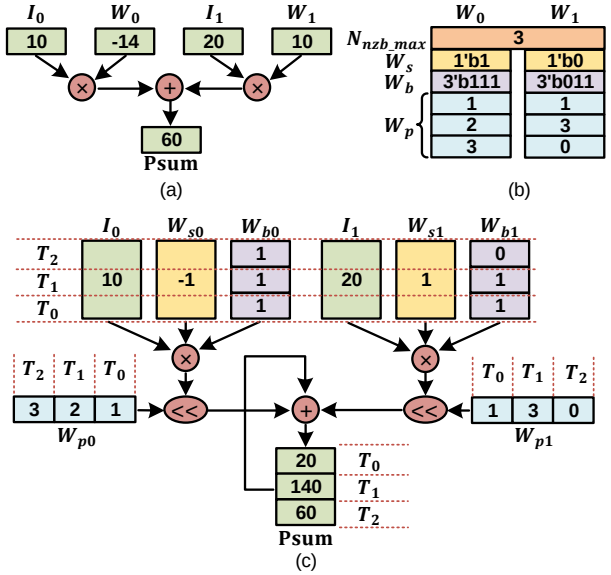

图 7：稀疏位串行计算示例。(a) 原始位并行计算。(b) 编码后的权重。(c) 稀疏位串行计算过程。

图 7 展示了稀疏位串行计算的一个示例。图 7(a) 展示了原始乘加运算，$Psum = I_0 * W_0 + I_1 * W_1$。图 7(b) 展示了两个权重的编码格式，其最大非零位数为 $N_{nzb_max} = 3$。计算流程如图 7(c) 所示。在 T_0 时，我们获取 $W_{p0} = 1$、$W_{p1} = 1$，然后 I_0 和 I_1 均左移 1 位，计算得到 $Psum = 20$。同样，在 T_1 时，部分和计算结果为 140。由于 W_{b1} 的最后一个位图为零，W_{p1} 的运算无效，最终结果为 60。整个过程耗时 3 个周期，比原始流程的 8 个周期快 2.67× 。

4 位平衡架构

为在硬件成本较低的情况下实现高加速比和高能效，我们选择脉动阵列 [24] 作为核心，以节省大量带宽并简化处理单元之间的连接。为进一步降低稀疏处理的开销，我们在软件层对权重进行编码，而不引入相应的预处理模块。稀疏位串行计算由顶层控制器根据最大非零位数进行控制。因此，我们架构中的处理单元只需要移位加法单元。整体架构的设计旨在以较低的硬件成本平衡稀疏权重位的工作负载，并与位稀疏量化协同工作。

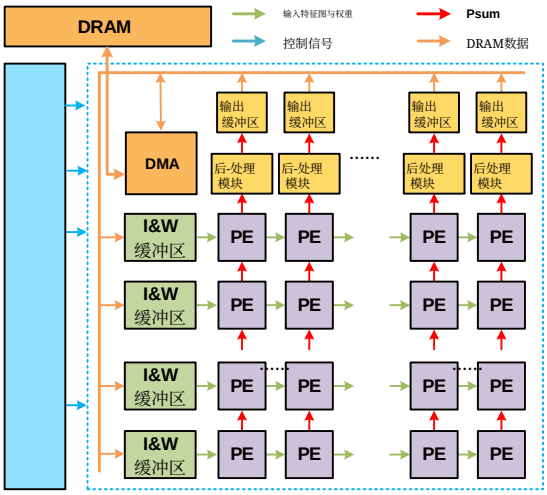

图 8：顶层架构。

4.1 概述

如图8所示，顶层架构主要由输入权重缓冲区（I&W 缓冲区）、基于脉动阵列的处理单元阵列、后处理模块、输出缓冲区、顶层控制器和 DMA接口组成。最初，在软件层生成编码格式的权重和控制参数。这些数据与输入图像被直接加载到动态随机存取存储器中。随后，输入与权重缓冲区获取输入特征图和权重，并将其传输至处理单元阵列。处理单元经过大量位串行运算后，后处理模块接收输出，并执行修正线性单元和池化操作，得到最终的输出特征图；这些输出特征图暂时存储在输出缓冲区中。最后，当前层的输出特征图

被存储到动态随机存取存储器中，用于下一层的计算。我们按层执行计算，并对最后一层的输出特征图进行分类，以输出最终结果。顶层控制器掌控整个计算流程，片上与片外数据通过 DMA接口传输。

4.2 基于脉动阵列的处理单元阵列

负责位串行计算的基于脉动阵列的处理单元阵列由 $N_{PE} \times N_{PE}$ 个处理单元组成。同一行的处理单元共享一个输入通道的输入特征图，同一列的处理单元则处理一个输出通道的输出特征图。每个处理单元包含一个补码处理单元、一个移位单元和一个累加单元，如图 9 所示。首先，根据权重符号 W_s，选择输入特征图的补码或原始值。然后，根据权重位位置 W_p 将临时结果左移。最后，将移位后的结果与相邻处理单元的结果累加。为降低功耗，当权重位图 W_b 为零时，补码处理单元和移位单元会被门控。此外，为适应输入特征图和权重的不同位宽，我们将 8 位运算与 16 位运算合并，以提高资源利用率。16 位输入特征图及其对应的 32 位临时结果可拆分为两组 8 位输入特征图和 16 位临时结果，因此，与 16 位运算相比，8 位运算可实现 $2\times$ 峰值吞吐量。

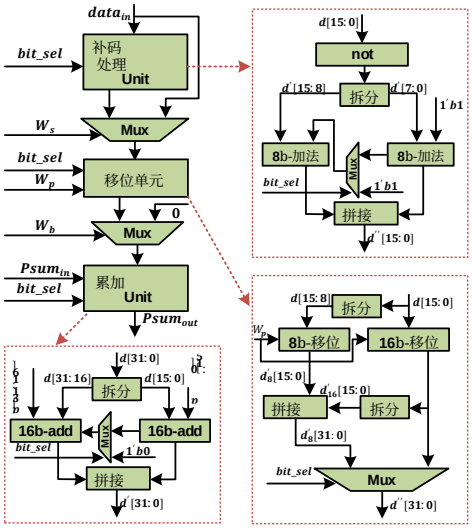

图 9：处理单元。

4.3 片上缓冲区

为了降低片上/片外数据移动的能耗，我们需要缓冲输入特征图、权重和输出特征图，以便重复利用数据。因此，片上缓冲区主要由输入与权重缓冲区和输出缓冲区组成。为避免数据加载造成性能瓶颈，我们将每两组存储器组成一个乒乓缓冲区，一组用于DRAM访问，另一组用于向处理单元传输数据，这与其他脉动阵列加速器类似 [25]。为支持自适应位宽计算，输入与权重缓冲区和输出缓冲区的位宽均为 16 位，可在每个地址存储一个 16 位输入特征图/输出特征图，或两个 8 位输入特征图/输出特征图。对于采用编码格式的权重，每个地址可存储 16 个权重符号、16 个权重位图、4 个 16 位精度的权重位置，或 5 个 8 位精度的权重位置。

5 稀疏位串行处理的数据流

为了降低DRAM访问的能耗，我们采用了一种数据流，以在有限的片上缓冲区资源下增强数据复用。由于 Bit-balance 是一种脉动阵列加速器，我们将稀疏位串行计算与脉动数据流相结合 [25]。此外，我们设计了一种映射算法，根据提供的网络参数将各种网络映射到我们的架构上。

5.1 数据划分

由于片上缓冲资源和处理单元阵列规模有限，我们需要根据边长以及输入/输出通道，将输入特征图和权重划分为独立的子瓦片，具体参见分块方法 [25]。考虑到部分和存储资源，输入特征图子瓦片的大小设定为不超过 $W_{IS} \times H_{IS}$，输入特征图瓦片在宽度和高度维度上的大小分别为 T_{WI} 和 T_{HI}。由于处理单元阵列规模有限，我们只能同时处理 N_{PE} 个输入和输出通道，分块数量分别为 T_{IC} 和 T_{OC}。具体分块参数如表 2 所示。数据划分后，每个输入特征图瓦片和权重瓦片均可独立进行稀疏位串行计算。

表 2：数据划分参数

参数	说明	公式
N_{PE}	处理单元阵列大小	-
N_{IC}	输入通道数	-
N_{OC}	输出通道数	-
W_K	卷积核宽度	-
H_K	卷积核高度	-
W_I	输入特征图宽度	-
H_I	输入特征图高度	-
W_{IS}	输入特征图块宽度	-
H_{IS}	输入特征图块高度	-
T_{IC}	输入通道分块数	N_{IC}/N_{PE}
T_{OC}	输入通道分块数	N_{OC}/N_{PE}
T_{WI}	IFM 宽度分块数	W_I/W_{IS}
T_{HI}	IFM 高度分块数量	H_I/H_{IS}

5.2 映射算法

考虑到数据流以及 IFM 和权重的分区，为了将各种网络映射到 Bit-balancing 上，我们设计了一种网络映射算法。计算流程的描述见表 3。第 1 行和第 4 行根据自适应数据流配置 [25]决定数据流。对于“优先复用 IFM（RIF）”，我们会先完成一个输出块中所有 OC 的 OFM 计算，再切换到下一个输出块；对于“优先复用权重（RWF）”，则会先完成一个 OC 的所有输出块计算，再切换到下一个 OC。第 2 行和第 3 行描述 IFM 的分区，其行数为 T_{ofm_row}，列数为 T_{ofm_col}。第 5 行控制输入通道的累加，以获得最终的部分和。第 6 行和第 7 行分别表示一个 IFM 块和一个卷积核中的元素数量。第 8 行表示权重的最大非零位数，其中 N_{nzb_max} 作为参数加载，因为训练得到的权重是固定的。第 9 行表示处理单元正在对 IFM 和权重位位置执行稀疏位串行计算。

表 3：计算流程

输入：架构配置的参数；输入特征图；权重；
输出：输出特征图

注意：当为 RIF 时，$T_{OC_RWF}=1$，$T_{OC_RIF}=Toc$；否则，$T_{OC_RWF}=Toc, T_{OC_RIF}=1$

```
for(a = 0; a < T_OC_RWF; a = a + 1)
  for(b = 0; b < T_WI; b = b + 1)
    for(c = 0; c < T_HI; c = c + 1)
      for(d = 0; f < T_OC_RIF; d = d + 1)
        for(e = 0; e < T_IC; e = e + 1)
          for(f = 0; f < W_K × H_K; f = f + 1)
            for(g = 0; g < W_K × H_K; g = g + 1)
              for (h = 0; h < N_nzb_max; h = h + 1)
                Psum+ = I_NZ × 2^W_p
```

6 实验

6.1 实现

Bit-balance 实现：为测量面积、功耗和关键路径延迟，我们使用 Verilog 实现了 Bit-balance，并在 Cadence Xcelium 2019.2 上进行了功能仿真。随后，使用 Synopsys Design Compiler（DC）对 Verilog 代码进行综合。片上 SRAM 缓冲器由 Memory Compiler 生成。片上功耗通过 Synopsys Prime Suite 2020.09，利用由测试平台生成的切换活动接口格式（SAIF）文件进行计算。资源利用率和功耗分解报告见表 4。

处理单元阵列的规模为 32×32，对于 16 位和 8 位移位加法运算，峰值吞吐量分别达到 $1024GOP/s$ 和 $2048GOP/s$ 。输入特征图和权重均采用 16 位或 8 位编码，其部分和相应地截断为 32 位和 16 位。由于处理单元采用移位加法单元实现，因此包含处理单元阵列、后处理模块和控制器的计算核心（CC）面积仅为 2.91mm^2。为平衡数据复用效率与资源开销，我们将输入特征图分块单元设为 8×8。每个输入与权重缓冲区包含两个 $1K \times 16b$ 和两个 $256 \times 16b$ 单端口静态随机存取存储器（SRAM）；每个输出缓冲区和后处理模块分别包含两个 $64 \times 16b$ 单端口寄存器文件（RF）和一个 $64 \times 16b$ 双端口寄存器文件。由此，片上存储器总计为 176KB。功耗主要来自计算核心，占整芯片功耗的 85%。

表 4：资源与功耗占比

模块	Area (mm2)	16b-功耗 (mW)	8b-功耗 (mW)
输入与权重缓冲区	1.81(36.3%)	90(11%)	86(10%)
处理单元阵列	2.34(46.9%)	582(71%)	609(71%)
后处理缓冲区	0.49(9.8%)	82(10%)	95(11%)
输出缓冲区	0.27(5.4%)	41(5%)	42(5%)
控制器	0.08(1.6%)	25(3%)	25(3%)
计算核心	2.91(58.3%)	689(84%)	729(85%)
总计	4.99	820	857

为验证 Bit-balance 的优越性，我们设置了表 5 所示的一些基线加速器。下面简要介绍各加速器：

1) Eyeriss [9] 是一种典型的基于16位定点乘加器的位并行加速器，不考虑输入特征图或权重的稀疏性。相比之下，Bit-balance通过利用权重位稀疏性实现更高的性能，并通过用移位加法单元替代乘加单元来实现更高的能效。

2) 寒武纪-X [13] 通过跳过零权重元素的计算来加速神经网络。然而，其整体架构是为加速 16 位精度的神经网络而设计的，因此在低位宽运算中资源利用效率较低。相比之下，Bit-balance 通过位串行计算实现了更高的资源效率。

3) Stripe [16] 允许逐层选择精度，而非使用定点值；基于位串行计算，其性能会随输入特征图的位宽成比例提升。但它只截断位宽，却忽略了输入特征图的位稀疏性。相比之下，Bit-balance 通过跳过权重中的零位实现了更高的性能。

4) Laconic [21] 将乘加运算拆分为输入特征图和权重的位串行计算，并通过利用输入特征图和权重的稀疏性实现显著加速。然而，输入特征图和权重中零位的分布可能不规则，从而导致处理单元阵列中的工作负载不均衡，降低性能。相比之下，Bit-balance 通过位稀疏量化实现了更高的性能。

5) Bitlets [23] 提出了一种位交织理念，以最大限度地利用位级稀疏性，通过减少参与计算的权重数量来实现加速。然而，位交织对应的逻辑占整个处理单元模块面积的 40%。此外，在低位运算期间，高位计算单元可能处于闲置状态，导致资源效率低下。相比之下，我们通过模型-硬件协同设计和自适应位宽计算，实现了更高的能效和资源效率。

在基准测试方面，我们在 AlexNet[1], VGG-16 [2], ResNet-50 [4], GoogleNet [3], Yolo-v3 [5]上进行了推理，并将我们的设计与各基线方案进行了比较。前四种神经网络在 ImageNet [1]上进行了测试，Yolo-v3 则在 CoCo [26]上进行了测试。

6.2 性能

为展示 Bit-balance 中稀疏处理方法的优势，我们将设计与 Eyeriss[9], 、寒武纪-X [13], 、Stripe [16], 、Laconic [21],和 Bitlets[23] 在归一化性能方面进行了比较，如图 10 所示，分别实现了 $1.6\times$~$8.6\times$、$1.1\times$~$2.4\times$、$4\times$~$7.1\times$、 $2.2\times$~$4.3\times$和 $1.1\times$~$1.9\times$ 的加速比。在 AlexNet、VGG-16、ResNet-50、GoogleNet 和 Yolo-v3 中，我们将最大非零位数 NNZB（ N_{nzb_max}）分别设为：16 位精度下的 3~4 ，以及 8 位精度下的 4~5 ，如表 6 所示。Top 1 和 Top 5 准确率两列展示了各神经网络的推理准确率，以及括号中对应的精度损失；这些数据参考了 PyTorch [27] 的训练结构。性能通过推理计算的频率与总周期之比计算；与 16 位精度下基础架构的位串行计算相比，加速比达到 $4\times$~$8\times$。尽管 16 位精度的准确率高于 8 位精度，且其最大非零位数 NNZB 也更小，但 8 位精度的峰值吞吐量是其两倍，因此自然能实现更高的性能。归一化性能（R_p）通过我们的性能与其他

表 5：各加速器的总体比较

加速器	工艺技术 (nm)	处理单元数量	Area (mm^2)	频率 (MHz)	峰值吞吐量 (GOP/s)	功耗 (mW)	分类
Eyeriss [9] JSSCC-2017	65nm	168	12.25@整芯片	200	33.6@16位乘累加器	278@AlexNet 234@VGG-16	位并行
寒武纪-X [13] MICRO-2016	65nm	16	6.38@整芯片	1000	256@16位乘累加器	278	稀疏 位并行
Stripe [16] CAL-2017	65nm	4096	122.1@整芯片	980	64225@16b 移位加法	-	位串行
Laconic [21] ISCA-2019	65nm	512	4.1@CC	1000	8192@1位移位加法	-	稀疏 位串行
Bitlet [23] MICRO-2021	65nm	32	5.80@CC	1000	768@24位移位加法	1390@16位 1199@ 8位	稀疏 位串行
Bit-balance	65nm	1024	4.99@整芯片 2.91@CC	1000	1024@16b 移位加法 2048@8b 移位加法	820@16b 857@ 8b	稀疏 位串行

表 6：各神经网络的输入特征图和权重稀疏率及精度损失

神经网络类型	16 位精度				8 位精度			
	N_{nzb_max}	性能 (帧/秒)	Top-1 准确率(%)	Top-5 准确率(%)	N_{nzb_max}	性能 (帧/秒)	Top-1 准确率(%)	Top-5 准确率(%)
AlexNet	3	270.5	55.6(-0.9)	78.8(-0.3)	5	326.2	55.6(-0.9)	78.8(-0.3)
VGG-16	3	20.4	70.8(-0.8)	89.9(-0.5)	4	30.1	71.2(-0.4)	90.1(-0.3)
GoogleNet	4	136.2	69.1(-0.7)	89.3(-0.3)	5	218.4	69.1(-0.7)	89.3(-0.3)
ResNet-50	3	46.8	75.2(-0.9)	92.4(-0.5)	5	56.3	74.9(-1.2)	92.4(-0.5)
Yolo-v3	3	10.9	61.9(-0.7)	-	4	16.4	60.3(-2.3)	-

加速器的性能之比计算。由于寒武纪-X、Stripe 和 Laconic 未在论文中提供其性能数据，因此归一化性能是基于在我们的基准测试中与 Bitlet 和 Eyeriss 的比较计算得出的。

与位并行架构相比，我们采用位串行计算，而非16位乘加运算单元。对于 Eyeriss，由于它已经流片，为了公平比较，我们假设 Eyeriss 的频率可达到 1吉赫兹（图 10 中的 Eyeriss-S）。尽管在一个处理单元中执行一次乘加运算需要多个周期，但我们的处理单元阵列规模大于 Eyeriss。此外，通过跳过零位，一次乘加运算的执行时间可缩短至 N_{nzb_max} 周期。在最佳情况下，我们在 VGG-16 上以 8位精度实现了 $30.1/(3.5) = 8.6\times$ 加速比。寒武纪-X 则利用权重元素稀疏性来加速计算。然而，每个处理单元所处理的权重稀疏性可能不均衡，从而导致性能下降。考虑到

通过同时利用稀疏性并调整处理单元阵列的大小，我们可以实现 1.1×~2.4× 性能提升。

与位串行架构相比，我们通过结合位稀疏量化来加速神经网络。我们在 Strip 中只对 NNZB 进行量化，而不是对位宽进行量化。因此，我们的量化方法能够适应更宽的权重值数值范围，并且 Bit-balance 中的 NNZB 小于 Stripe 中的位宽。因此，我们获得了 $4\times$~$7.1\times$ 加速比。与 Laconic 相比，我们通过将 NNZB 限制为某个值，在处理单元阵列中实现了工作负载均衡。尽管 Laconic 利用了输入特征图和权重的稀疏性，但其计算时间由最长的非零位序列决定，因此一旦计算中出现很长的位序列，性能就会下降。通过负载均衡，我们实现了 $2.2\times$~$4.3\times$ 性能提升。对于 Bitlet，在 16 位精度下，位交织带来的性能提升与我们的方法相近。但我们采用自适应数据位宽计算，以在低位运算中实现更高性能；而在 Bitlet 中，一些计算单元仍处于空闲状态。在最佳情况下，我们在 ResNet-50 的 8位精度下获得了 $56.3/29 = 1.9\times$ 加速比。

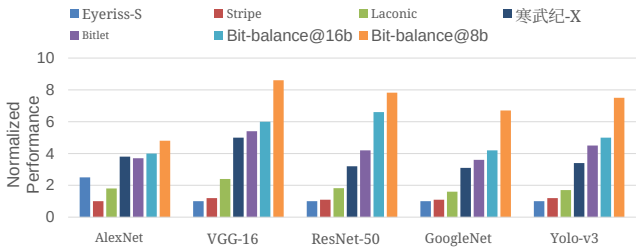

图 10：与 Eyeriss、寒武纪-X、Stripe、Laconic 和 Bitlets 的性能比较。

6.3 能效与资源效率

由于各加速器采用的处理方法各不相同，因此有必要探究其收益是否超过由此产生的开销。我们比较了 Bit-balance 与 Eyeriss [9], 、寒武纪-X [13], 、Stripe [16], 、Laconic [21], 和 Bitlet [23],的能效与资源效率。如图 11 所示，它们的能效分别提升了 $2.7\times$~$11.3\times$、$1.3\times$~$2.8\times$、$3\times$~$5.6\times$、$2.7\times$~$5.4\times$和 $1.8\times$~$2.7\times$；如图 12 所示，它们的资源效率分别提升了 $3.9\times$~$21\times$、$1.6\times$~$3.9\times$、$1.7\times$~$3\times$、$3.2\times$~$6.3\times$和 $2.1\times$~$3.8\times$。能效的计算方式为 R_{np}/R_p，其中 R_{np}和 R_p 分别表示 Bit-balance 与其他基线在归一化性能和功耗方面的比值。对于资源效率，在能效表达式中以面积比 R_a替换功耗比 R_p。由于 Stripe 的处理单元阵列大小已针对归一化性能计算进行缩放，因此表达式 R_{np}/R_a 还应乘以峰值性能比 R_{pp}，以表示单位面积吞吐量。

与 定点架构相比，MA C 个单元

在 Bit-balance 中，C 单元的面积和功耗高于加法移位单元。对于 Eyeriss，其 PE 由一个乘加单元和一些存储器组成，其中乘加单元仅占总面积和总功耗的 9%，这表明与 Bit-balance 相比，Eyeriss 为每次乘加运算消耗了更多资源和能量。需要注意的是，我们将 Eyeriss 的频率调整至 1吉赫兹，功耗应按 $5\times$ 进行缩放，而面积保持不变。在 VGG-16 的 8位精度下，最佳情况下，我们实现了 $8.6/(857/(236\times 5))=13.4\times$能效和 $8.6/(4.99/12.25)=21\times$ 资源效率提升。对于寒武纪-X，它引入了用于稀疏权重处理的索引模块，占总面积的 31% 和总功耗的 34%。然而，Bit-balance 中的稀疏权重处理与密集位串行计算类似，因此开销很小。因此，我们实现了$1.3\times$~$2.8\times$ 能效和 $1.6\times$~$3.9\times$ 资源效率提升。

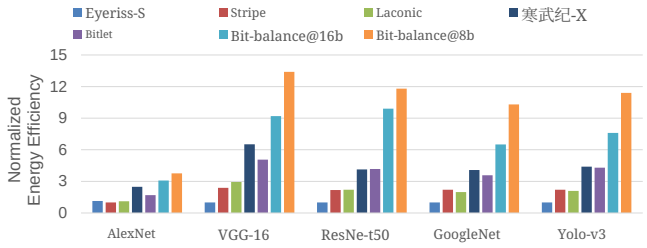

图 11：与 Eyeriss、寒武纪-X、Stripe、Laconic 和 Bitlets 的能效比较。

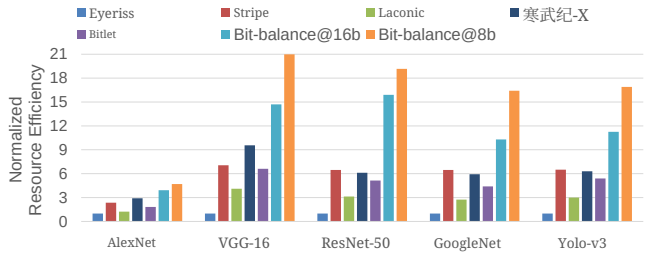

图 12：与 Eyeriss、寒武纪-X、Stripe、Laconic 和 Bitlets 的资源效率对比。

与位串行架构相比，Bit-balance 中稀疏计算的开销更小。对于 Stripe，它将 16 个移位加法单元合并为一个模块，简化了中间步骤；而 Bit-balance 则在每个处理单元中分别执行移位加法运算以进行稀疏处理。此外，由于处理单元阵列的规模按平方增长，而存储器的规模按线性增长，因此处理单元阵列规模越大，存储器所占比例就越小。因此，在执行一次移位加法运算时，其资源消耗$(64225/1024)/(122.5/4.99)=2.5\times$ 低于 Bit-balance。综合考虑开销与性能，我们实现了 $3\times$~$5.6\times$ 能效和 $1.7\times$~$3\times$ 资源效率提升。对于 Laconic，其处理单元将乘加运算分解为 16 对输入特征图位和权重位运算。尽管它能够利用输入特征图位和权重位的稀疏性，但仍有许多逻辑单元用于处理中间结果，因此与 Bit-balance 相比，计算核心面积开销增加了 $4.1/2.91=1.4\times$ 。因此，我们实现了$2.7\times$~$5.4\times$ 能效和 $3.2\times$~$6.3\times$ 资源效率提升。对于 Bitlet，其计算过程包含三个步骤：预处理、动态指数匹配和位提炼。每个步骤都由相应的模块实现，因此增加了 45% 的面积，并

功耗开销为 69%。不过，Bit-balance 中的权重预处理在线下执行，不会产生逻辑开销。跳过零位的过程则由顶层控制器模块掌控，无需在处理单元中引入稀疏处理逻辑。因此，在最佳情况下，我们在 ResNet-50 的 8 位精度下实现了 $1.9/(857/1199)=2.7\times$ 能效和$1.9/(2.91/5.80)=3.8\times$ 资源效率提升。

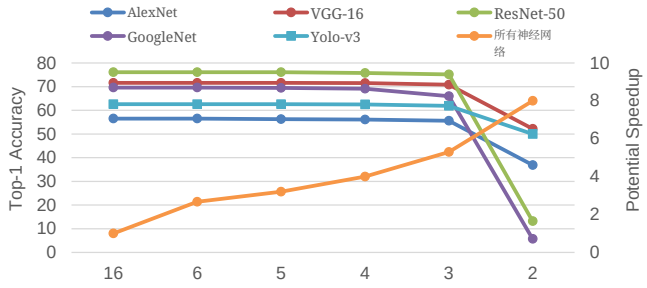

图 13：16 位精度下，不同N_{nzb_max} 对应的潜在加速比和精度损失。

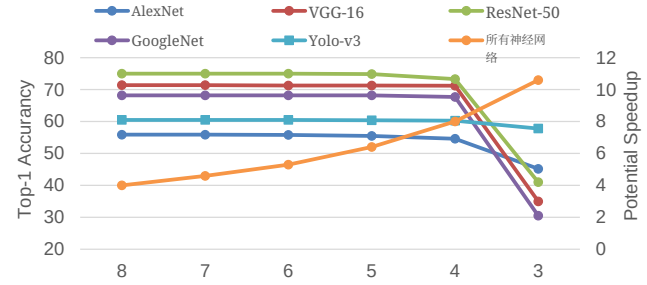

图 14：8 位精度下，不同N_{nzb_max} 对应的潜在加速比和精度损失。

6.4 对稀疏性的敏感度

权重位的稀疏性越高，性能越高，但会牺牲准确率。为了探究权重位稀疏性对性能和准确率的影响，我们训练了几组具有不同最大非零位数的权重，如图 13 和图 14 所示。根据 Pytorch [27]，AlexNet、VGG-16、ResNet-50、GoogleNet 和 Yolo-v3 在 32 位浮点精度下的 top-1 准确率分别为 56.5% 、71.6%、76.1%、69.8% 和 62.6%。对于 16 位精度，在不进行位稀疏性量化的情况下，各神经网络的精度损失仅为 0~0.2%。随后，我们将最大非零位数进行量化为 2~6，实现了 $2.67\times$~$8\times$ 加速比。在 $N_{nzb_max}=6$~4阶段，所有神经网络的准确率保持稳定，仅损失 0.1%~1.0%。在 [27]时，

GoogleNet 的准确率下降了约 3.7%。当 $N_{nzb_max} = 2$时，所有神经网络的准确率都会骤降。因此，在我们的工作中，对于 16 位精度，我们选择了 $N_{nzb_max} = 3$ 或 4 。

8 位精度下的现象与 16 位精度类似。未采用位稀疏量化时，各神经网络在 8 位精度下的精度损失为 0.2%~2.1%。在 $N_{nzb_max} = 7$~5阶段，准确率保持稳定。在 $N_{nzb_max} = 4$时，AlexNet、ResNet-50 和 GoogleNet 的准确率下降约 1.9%~2.8%。当 $N_{nzb_max} = 3$时，所有神经网络的准确率都会急剧下降。因此，对于 8 位精度计算，我们选择了 $N_{nzb_max} = 4$或 5 。

尽管由于自适应位宽计算，8 位精度的性能提升高于 16 位精度下的 Bit-balance，但仍有一些特定领域的神经网络更适合采用 16 位精度来保持准确率。例如，Yolo-v3 在 16 位精度下的准确率平均比 8 位精度高 2%，这对神经网络而言是显著的准确率提升。因此，自适应位宽对于神经网络加速中的稀疏位串行计算至关重要。

6.5 与基本位串行架构的比较

由于 Bit-balance 采用编码格式处理权重，与原始权重相比，会带来存储开销并增加 DRAM 访问的功耗。为了说明 Bit-balance 中稀疏处理的优势，我们比较了整个系统与基本位串行架构（不进行稀疏处理的 Bit-balance）的能效。DRAM 访问的功耗根据 DRAM 访问总次数和运行时间，使用 CACTI [28] 进行估算。基于第 V 节中的数据流，尽管如图 15 和图 16 所示，我们在 $1\times$~$2.4\times$ DRAM 访问上消耗了 $1\times$~$1.4\times$ 功耗，但由于性能提升，如图 17 所示，Bit-balance 的能效仍比基本位串行架构高 $1.14\times$~$5.3\times$ 。

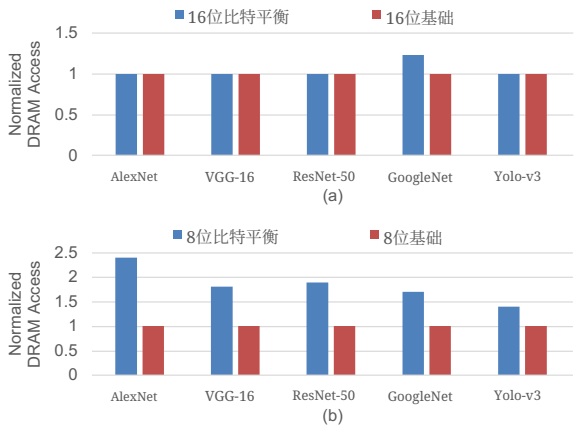

图 15：与基本位串行架构的 DRAM 访问对比。(a) 16 位精度。(b) 8 位精度。

对于 16 位精度，使用 $N_{nzb_max} = 3$ 的编码格式每个权重占用 16 位，其中 1 位用于符号位，3 位用于位图，3×4 位用于权重位置。在 $N_{nzb_max} = 4$ 的情况下，每个权重占用 $1 + 4 + 4 \times 4 = 21$ 位。由于与原始 16 位权重相比，该编码格式会产生 $1\times$~$1.3\times$ 权重存储开销，因此 DRAM 访问量会增加 $1\times$~$1.23\times$。因此，Bit-balance 的功耗为 $1\times$~$1.07\times$ 。由于 $4\times$~$5.3\times$ 性能，Bit-balance 的能效比基本位串行架构 $3.73\times$~$5.3\times$ 高。

对于 8 位精度，使用$N_{nzb_max} = 4$ 和 $N_{nzb_max} = 5$ 时，每个权重的存储空间分别为 17 位和 21 位，与原始 8 位权重相比，权重存储开销为 $2.1\times$~$2.6\times$ 。因此，Bit-balance 为 $1.4\times$~$2.4\times$ DRAM访问消耗的功耗为 $1.12\times$~$1.41\times$ 。由于 $1.6\times$~$2\times$ 性能，Bit-balance 的能效比基本位串行架构 $1.14\times$~$1.79\times$ 高。

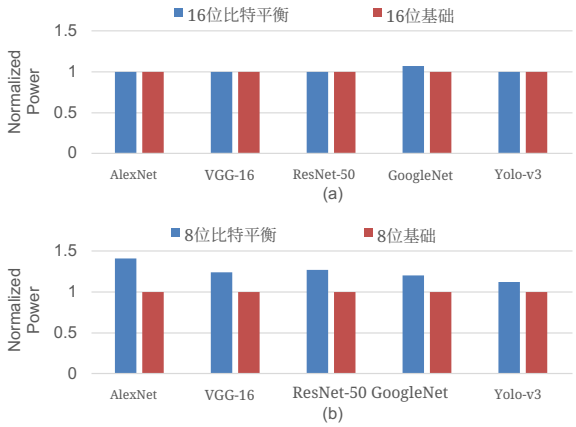

图 16：与基本位串行架构的功耗比较。(a) 16 位精度。(b) 8位精度。

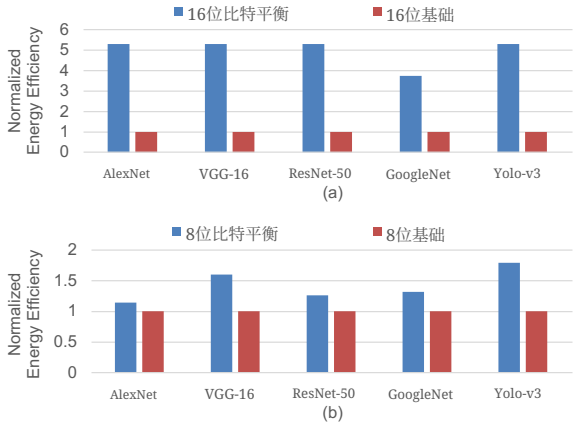

图 17：与基本位串行架构的能效比较。(a) 16 位精度。(b) 8位精度。

7 结论

本文提出了一种用于稀疏权重位处理的稀疏位串行加速器 Bit-balance，以较低的硬件成本实现了显著的性能提升。同时，我们协同设计了一种位稀疏量化方法，在精度损失较小的情况下，将权重的最大非零位数控制在某一特定值以内，从而

有效平衡处理单元阵列中稀疏权重位的工作负载。此外，为支持自适应位宽计算，我们将 8 位精度与 16 位精度运算相结合，以提高资源效率。与以往的稀疏加速器相比，Bit-balance 能以更小的面积实现更高的性能和能效。

参考文献

[1] A. Krizhevsky、I. Sutskever 和 G. E. Hinton，《使用深度卷积神经网络进行 ImageNet 分类》，《Commun. ACM》，第 60 卷，第 6 期，第 84–90 页，2017 年 5 月。 [在线]。网址：https://doi.org/10 1145 3065386 。/ K. Simonyan 和 A. [2] Zisserman，《用于大规模图像识别的超深卷积网络》，2015 年。[3] C. Szegedy、W. Liu、Y. Jia 等，《通过卷积走向更深层》，载于 2015 年 IEEE 计算机视觉与模式识别会议 (CVPR)，2015 年，第 1–9 页。[4] K. He、X. Zhang、S. Ren 等，《用于图像识别的深度残差学习》，载于 2016年 IEEE 计算机视觉与模式识别会议 (CVPR)，2016 年，第 770–778 页。[5] J. Redmon、S. Divvala、R. Girshick 等，《只看一次：统一的实时目标检测》，载于 2016年 IEEE 计算机视觉与模式识别会议 (CVPR)，2016 年，第 779–788 页。[6] T. N. Sainath、O. Vinyals、A. Senior 等，《卷积、长短期记忆与全连接深度神经网络》，载于 2015年 IEEE 国际声学、语音与信号处理会议 (ICASSP)，2015 年，第 4580–4584 页。[7] Y. Jia、E. Shelhamer、J. Donahue、S. Karayev 等，《Caffe：用于快速特征嵌入的卷积架构》，载于第22届 ACM 多媒体国际会议论文集，丛书 MM '14。美国纽约州纽约市：计算机协会，2014 年，第 675–678 页。 [在线]。网址：https://doi.org/10 1145 2647868 2654889 。/ . Y. Chen、T. Luo、[8] S. Liu、S. Zhang、L. He、J. Wang、L. Li、T. Chen、Z. Xu、N. Sun 和 O. Temam，《Dadiannao：一台机器学习超级计算机》，载于 2014年第47届 IEEE/ACM 微体系结构国际研讨会，2014 年，第 609–622 页。[9] Y.-H. Chen、T. Krishna、J. S. Emer 等，《Eyeriss：用于深度卷积神经网络的高能效可重构加速器》，《IEEE 固态电路期刊》，第 52 卷，第 1 期，第 127–138 页，2017 年。[10] N. P. Jouppi、C. Young、N. Patil 等，《张量处理单元在数据中心中的性能分析》，载于 2017 年第 44 届 ACM/IEEE 计算机体系结构国际研讨会 (ISCA)，2017 年，第 1–12 页。[11] L. Deng、G. Wang、G. Li 等，《天机：连接脉冲式与连续神经计算的统一可扩展芯片》，《IEEE 固态电路期刊》，第 55 卷，第 8 期，第 2228–2246 页，2020 年。[12] S. Han、X. Liu、H. Mao 等，《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 计算机体系结构国际研讨会 (ISCA)，2016 年，第 243–254 页。[13] S. Zhang、Z. Du、L. Zhang 等，《寒武纪-X：稀疏神经网络加速器》，载于 2016年第49届 IEEE/ACM 微体系结构国际研讨会 (MICRO)，2016 年，第 1–12 页。[14] J. Albericio、P. Judd、T. Hetherington 等，《Cnvlutin：无需无效神经元的深度神经网络计算》，载于 2016年第43届 ACM/IEEE 计算机体系结构国际研讨会 (ISCA)，2016 年，第 1–13 页。[15] A. Parashar、M. Rhu、A. Mukkara 等，《SCNN：压缩稀疏卷积神经网络加速器》，载于 2017年第44届 ACM/IEEE 计算机体系结构国际研讨会 (ISCA)，2017 年，第 27–40 页。[16] P. Judd、J. Albericio 和 A. Moshovos，《Stripes：位串行深度神经网络计算》，《IEEE 计算机体系结构快报》，第 16 卷，第 1 期，第 80–83 页，2017 年。[17] S. Sharify、A. D. Lascorz、K. Siu、P. Judd 和 A. Moshovos，《Loom：利用权重与激活值精度加速卷积神经网络》，载于 2018年第55届 ACM/ESDA/IEEE 设计自动化会议 (DAC)，2018 年，第 1–6 页。

[18] H. Sharma、J. Park、N. Suda、L. Lai、B. Chau、V. Chandra 和 H. Esmaeilzadeh，《Bit fusion：用于加速深度神经网络的位级动态可组合架构》，载于*2018*年*ACM/IEEE*第*45*届计算机体系结构国际研讨会 *(ISCA)*，2018 年，第 764–775 页。[19] J. Lee、C. Kim、S. Kang、D. Shin、S. Kim 和 H.-J. Yoo，《Unpu：具有完全可变权重位精度的高能效深度神经网络加速器》，*IEEE* 固态电路期刊，第 54 卷，第 1 期，第 173–185 页，2019 年。[20] J. Albericio、A. Delmás、P. Judd、S. Sharify、G. O' Leary、R. Genov 和 A. Moshovos，《Bit-pragmatic 深度神经网络计算》，载于*2017*年第*50*届*IEEE/ACM*微体系结构国际研讨会 *(MICRO)*，2017 年，第 382–394 页。[21] S. Sharify、A. D. Lascorz、M. Mahmoud、M. Nikolic、K. Siu、D. M. Stuart、Z. Poulos 和 A. Moshovos，《Laconic 深度学习推理加速》，载于*2019*年*ACM/IEEE*第*46*届计算机体系结构国际研讨会 *(ISCA)*，2019 年，第 304–317 页。[22] A. Delmas Lascorz、P. Judd、D. M. Stuart、Z. Poulos、M. Mahmoud、S. Sharify、M. Nikolic、K. Siu 和 A. Moshovos，《Bit-tactical：利用神经网络中的数值与位稀疏性的软件/硬件方法》，载于第二十四届编程语言与操作系统架构支持国际会议论文集，丛书 ASPLOS '19。美国纽约州纽约市：计算机协会，2019 年，第 749–763 页。

[ ] 10 1145 3297858 3304041 在线。[23] 网址：https://doi.org/ 。/ . H. Lu、L. Chang、C. Li、Z. Zhu、S. Lu、Y. Liu 和 M. Zhang，《提炼位级稀疏并行性，以实现通用深度学习加速》，载于*MICRO-54*：第*54*届*IEEE/ACM*微体系结构国际研讨会，丛书 MICRO '21。美国纽约州纽约市：计算机协会，2021 年，第 963–976 页。 [在线]。网址：https://doi.org/10 1145 3466752 3480123 。/ . Kung，[24] 《为什么采用脉动架构？》《*Computer*》，第 15 卷，第 1 期，第 37–46 页，1982 年。[25] W. Sun，《Sense：用于在脉动阵列上加速稀疏卷积神经网络的模型与硬件协同设计》，https://arxiv.org/abs/2202.00389。[26] 《COCO数据集》，https://cocodataset.org/#download。[27] M. Paganini，《剪枝教程》，https://pytorch.org/tutorials/ intermediate/pruning_tutorial.html。[28] R. Balasubramonian、A. B. Kahng、N. Muralimanohar 等，《Cacti 7：用于创新型片外存储器互连探索的新工具》，《*ACM*体系结构与代码优化汇刊》，第 14 卷，第 2 期，2017 年 6 月。 [在线]。网址：https://doi.org/10.1145/3085572