doi:10.11887/j.cn.202305025 http://journal.nudt.edu.cn

稀疏卷积计算高效数据加载与输出缓存策略*

刘 彪,陈长林,张宇飞,刘思彤,唐励勤,于红旗
(国防科技大学 电子科学学院,湖南 长沙 410073)

摘 要:针对现有神经网络加速器在处理稀疏神经网络时存在的数据加载效率低、乘加资源利用率低、输出缓存寻址逻辑复杂等问题,提出了稀疏卷积计算高效数据加载与输出缓存策略。将属于同一输入通道的非零输入特征图像数据和非零权重进行全对全乘累加运算,降低了非零数据配对难度,提高了乘加资源利用率;通过采用输入驻留计算,以及密集型循环加载特征图像数据,大幅减少了数据片外调取次数;优化了输出缓存设计,解决了现有方案中存在的输出缓存地址访问争用、存储拥塞等问题。实验表明,与采用类似架构的细粒度脉动加速器相比,在处理单元面积上减少了21.45%;在数据加载速度方面平均提高了117.71%;在平均乘法器利用率方面提高了11.25%,达到89%。

关键词:神经网络加速器;稀疏卷积神经网络;输入驻留;全对全计算
中图分类号:TN492 **文献标志码**:A **开放科学(资源服务)标识码(OSID)**:
文章编号:1001-2486(2023)05-212-10

High-efficiency data loading and output buffering strategy for sparse convolutional computing

LIU Biao, *CHEN Changlin*, *ZHANG Yufei*, *LIU Sitong*, *TANG Liqin*, *YU Hongqi*
(College of Electronic Science and Technology, National University of Defense Technology, Changsha 410073, China)

Abstract: In view of the problems such as inefficient data loading, insufficient utilization of multiply-accumulates resources, complex output buffering and addressing logic in existing neural network accelerators when processing sparse neural networks, a high-efficiency data loading and output buffering strategy for sparse convolutional computing was proposed. It performed an all-to-all multiply-accumulates operation on the non-zero input feature map data and the non-zero weights belonging to the same input channel, which reduces the difficulty of non-zero data pairing and improves the utilization of multiply-accumulates resources. By using input stationary calculation and intensive cyclic loading of input feature map data, it significantly reduced the number of data off-chip fetches. It optimized the output buffer design and solved the problems of address access contention and storage congestion during output buffering in existing solutions. Experimental results show that, when compare to fine-grained systolic accelerator with similar architectures, the process element area of the proposed architecture is decreased by 21.45%; the data loading speed is increased by 117.71% on average; the average utilization of multiplier is increased by 11.25%, reaching 89%.

Keywords: neural network accelerator; sparse convolution neural network; input stationary; all-to-all calculation

为实现更强的识别能力和更高的识别精度,现代深度神经网络(deep neural network, DNN)规模不断增大,在推理过程中也需要更加庞大的计算量和存储空间。然而通过深入研究深度神经网络权重矩阵和各层特征图像数据结构可以发现,神经网络内部存在较大的稀疏度,即有较大比例的权重系数和神经元输出为0值[1-2]。DNN的稀疏特性可以分为动态稀疏和静态稀疏[3]:动态稀疏是由于部分神经元输出经激活后变为零值而产生的,这种稀疏程度随神经元的输入变化而变化;静态稀疏又包括神经元稀疏和权重稀疏,二者在进行网络训练时,因不影响正确的网络识别结果而被删除。在神经网络加速器中,增加对稀疏计算的支持,能够大幅减少无效数据存储和计算,使得推理计算效率得到大幅提升[4-6]。

近年来,为高效执行稀疏DNN,涌现了大量采用定制化架构的稀疏DNN加速器[5-12]。通过向乘加阵列加载输入特征图像(input feature map, IFM)数据和权重前剔除0值数据并实现二者的正确配对,这些加速器在计算资源利用效率

* **收稿日期**:2022-06-08
基金项目:国家自然科学基金资助项目(61804181,62074166);国家重点研发计划资助项目(2019YFB2205102)
作者简介:刘彪(1998—),男,湖南娄底人,硕士研究生,E-mail:liubiao_nudt@163.com;
于红旗(通信作者),男,河南开封人,副教授,博士,硕士生导师,E-mail:13755132901@163.com

和推理计算速度方面均获得大幅提升。然而现有稀疏神经网络加速器仍然存在以下问题需要解决：①在加载输入数据时，需引入有效数据配对等额外操作，增加了硬件资源设计复杂度，为加速器带来额外面积和功耗负担；②在推理计算时，因配对效率较低而导致大量乘加资源处于空闲状态，无法充分利用资源加速推理计算；③在存储输出数据时，仲裁单元和输出缓存单元过于庞大且单一，在面临较多输出数据时，容易造成数据拥塞问题。为解决上述问题，进一步优化稀疏 DNN 数据处理方式，本文提出了稀疏卷积计算高效数据加载与输出缓存策略，主要包括：

1）采用全对全数据流计算，将属于同一输入通道的非零 IFM 数据与非零权重进行任意两两配对计算，降低了非零数据配对难度，提高了乘加资源利用率。

2）采用输入驻留计算，以及密集型循环加载 IFM 数据，大幅减少了数据片外调取次数。

3）优化输出缓存设计，简化乘累加中间结果缓存寻址难度，实现了输出数据的简便、快速存储，避免了数据访问争用、存储拥塞等问题。

1　DNN 加速器现状

稀疏 IFM 数据和稀疏权重导致 DNN 推理计算过程中存在大量的无效数据存储和计算。图 1 展示了稀疏卷积计算特点，理论计算包含了较多 0 值数据所参与的无效计算，实际只需较少的有效计算便可得到结果，而且 0 值数据的存储也会造成资源和能量消耗。

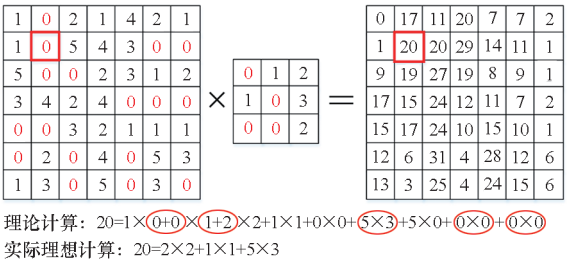


图 1　稀疏卷积计算特点

Fig. 1　Sparse convolutional calculation characteristic

针对上述问题，为提高处理稀疏网络的性能和效率，研究人员设计了多种稀疏 DNN 加速器。例如，针对特征图像中存在的稀疏性而被设计的加速计算架构 Cnvlutin[5]、Cambricon-X[6] 等。这类架构利用单索引或者偏移量对特征图像数据进行筛选配对，随后配对数据被输送至乘法器完成计算，再由加法树实现累加。由于网络的稀疏，需要配对的有效数据较少，如要使配对数据填满计算资源以提高计算效率，就需要复杂的配对及控制逻辑，存在较高的设计难度。Li[7]、Lu[8] 等针对权重矩阵中的稀疏性进行了定制设计：首先将单个权重需要计算的 IFM 数据进行全部提取，然后该权重与所有 IFM 数据乘积得到一个中间结果图像，最后对所有权重进行同样操作并累加中间结果图像得到完整的输出特征图像（output feature map，OFM）。但在处理浅层 DNN 时，需要使用较多的缓存空间以提取全部 IFM 数据，同时也需暂存每个中间结果图像，需要较多硬件存储资源。Liu 等[9] 通过判断乘法器所加载的 IFM 数据是否为 0 以控制计算的执行，可以阻止无效计算的产生并减少计算功耗，但未执行计算的资源处于空闲，存在利用不充分的现象。同时该方法在操作上仅避免了无效计算并没有节省计算周期，无法有效加速推理计算。

可以看到，上述加速器由于只考虑到了 DNN 中的一种稀疏特性，仍存在无效数据的加载和计算，计算效率提升不够显著，数据流和硬件架构也需优化。更多设计则把特征图像中动态稀疏和静态稀疏均纳入设计考量。

Zhou[3]、Lin[10] 等针对 IFM 数据和权重数据稀疏，设计了双索引逻辑实现有效数据配对，这种配对有效数据的方式与 Cambricon-X 相似，不可避免地给加速器设计增添了复杂度和额外功耗。VSCNN[11] 通过将输出数据的中间结果依次传输给不同处理单元（processing element，PE）以加速输出数据累加，并通过剔除 IFM 上规则分布的 0 值数据，实现稀疏网络处理优化，但无法加速不规则网络。SCNN[12] 直接输入 IFM 数据和权重高效完成卷积计算而不是采取数据配对的方式，并根据输出数据的 OFM 坐标将数据存储在输出缓存单元，但在存储输出数据时会面临同一地址有多个数据存储的争用问题。文献[13]设计了一个针对稀疏卷积神经网络的细粒度脉动加速器（fine-grained systolic accelerator，FSA），利用 IFM 数据沿乘法器阵列水平方向以脉动的方式移动，实现 IFM 数据复用。这种脉动方式需花费一定周期才能让 IFM 数据遍历所有乘法器，在 IFM 数据量较少时，IFM 只需水平移动较少的周期便完成了计算，会导致位于脉动方向较后的乘法器无法接收数据，因此存在乘法器利用率低的问题。同时，FSA 与多数加速器相似，在利用输出数据的 OFM 坐标为索引存储数据时，需将所有推理计算结果输送至同一个仲裁单元完成数据分配。该仲裁单元受密集数据的影响，会发生数据拥塞，并不利于数据的高效存储，

且在设计上需要较大的内部数据带宽,导致该单元在加速器中有较大的面积占比。

综上所述,现有加速器能在稀疏 DNN 上实现加速,但存在数据流低效、资源利用率不高、硬件设计复杂等问题,因此亟须进一步优化稀疏 DNN 加速器设计。

2　高效的数据加载和输出缓存设计实现

为了有效克服上述挑战,使加速器获取更好的处理性能,提出了稀疏卷积计算高效数据加载与输出缓存策略,主要研究工作如下。

2.1　全对全数据流计算

如图 2 所示,DNN 的计算模式表现为:编号为 C_o 的卷积核权重矩阵与输入特征图像矩阵计算时,相同输入通道 C_i 的权重矩阵与输入特征图像矩阵进行卷积,得到该输入通道的中间结果图像,然后将所有输入通道的中间结果对应相加,得到输出通道 C_o 的完整输出特征图像。

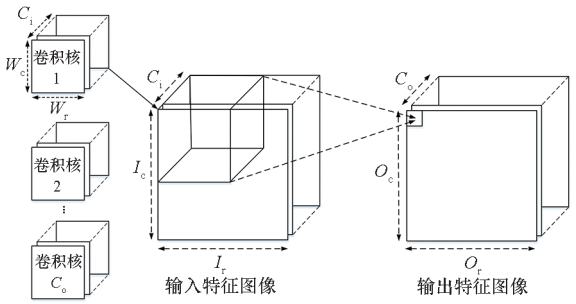


图 2　DNN 计算示意图

Fig. 2　DNN calculation schematic

通过分析同一输入通道数据的卷积运算可知,除位于 IFM 边缘的数据只需与卷积核中部分权重进行乘积外,IFM 中的大部分数据需要与该输入通道内的所有权重进行乘积。因此为了提高计算效率,本设计选择采用全对全的方式进行卷积运算,亦即,将属于同一输入通道的所有 IFM 数据与所有权重均进行相乘运算,同时根据二者坐标计算输出坐标,根据输出坐标值确定计算结果是否有意义。但这一计算方式也会导致边缘 IFM 数据引入部分非必要的冗余计算。冗余计算比例可用乘法器利用率进行衡量:

$$\text{乘法器利用率} = \frac{\text{计算总量} - \text{冗余计算量}}{\text{计算总量}} \quad (1)$$

计算总量是 IFM 非零数据总数和非零权重总数相乘得出;冗余计算量则是边缘 IFM 数据多余的计算量。图 3 展示了在不同 IFM 稀疏度和 IFM 尺寸下全对全乘积的乘法器利用率。可以看到,在 IFM 不存在稀疏的情况下,IFM 尺寸大于 14 像素 × 14 像素时,全对全数据流计算的乘法器利用率均高于 90%。虽然在 IFM 尺寸较小时,随着计算总量的减少,冗余计算的占比有所增加,导致乘法器利用率下降,但在实际情况下,受稀疏的影响,位于 IFM 边缘的数据量有所减少,计算总量中冗余计算的占比也因此下降,故乘法器利用率得以提升。在图 3 中表现为乘法器利用率随着 IFM 稀疏度的增加而增加,因此 IFM 数据的稀疏有利于全对全数据流计算。虽然引入的冗余计算会降低乘法器利用率,但对整体计算的影响并不严重。

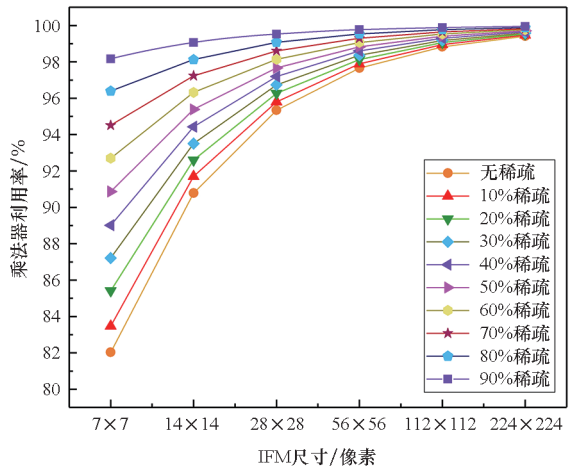


图 3　不同 IFM 稀疏度和 IFM 尺寸下乘法器利用率

Fig. 3　Multiplier utilization for different IFM sparsity and IFM size

为实现全对全数据流计算,设计了如图 4 所示的 $N \times M$ 乘加阵列,其中 N 和 M 均应为 2 的整数次幂以简化寻址逻辑。以 $N=8, M=8$ 为例介绍乘加阵列工作过程:乘加阵列的上方接收权重,

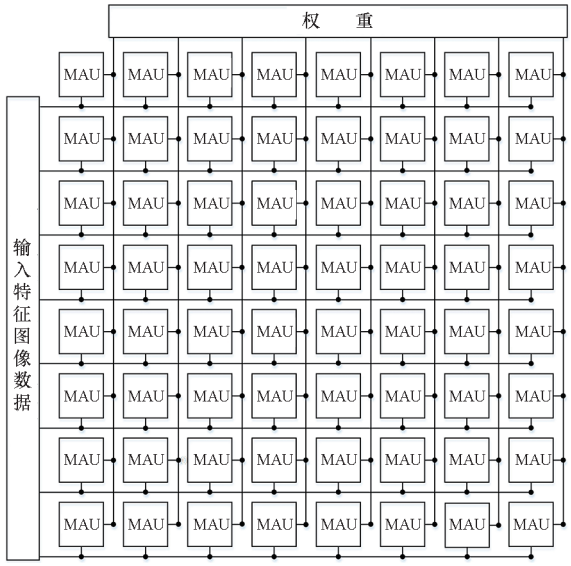


图 4　乘加阵列

Fig. 4　Multiplier and accumulator array

每列共享相同权重;左方接收 IFM 数据,每行共享相同 IFM 数据。在进行推理计算时,只要属于相同输入通道的 IFM 数据和权重即需要两两乘积,全对全数据流可以在每个计算周期获取 64 个不同的乘积结果。

累加并存储输出数据的中间结果时,本设计同步为每个乘加单元(multiply and accumulate unit, MAU)搭配一个独立的地址计算单元。该单元根据 IFM 数据的行、列坐标以及权重的行、列坐标实时地计算出输出数据的 OFM 坐标,根据 OFM 坐标将输出数据存储在输出缓存单元的不同地址,OFM 坐标计算公式[13]如下:

$$O_r = I_r + \lfloor \frac{KS}{2} \rfloor - W_r \tag{2}$$

$$O_c = I_c + \lfloor \frac{KS}{2} \rfloor - W_c \tag{3}$$

其中:O、I、W 分别表示输出、IFM 数据和权重;下标 r、c 分别表示行坐标和列坐标;KS 表示卷积核窗口尺寸。如果计算得到的 OFM 坐标超出了 OFM 实际尺寸范围,则标记该输出数据为无效以屏蔽其后续的累加和存储等操作。

2.2　输入数据加载与复用

为节约存储空间,IFM 数据和权重矩阵通常采用稀疏编码格式进行存储,如按照原始存储顺序加载 IFM 数据和权重,在形式上虽能得到所有推理计算的中间结果,但卷积计算结果输出顺序将会变得混乱。比如,压缩后的数据虽然密集但无规律,在 IFM 数据与权重相乘时,所有乘积结果分布在输出特征图像的各个位置,从而导致缓存寻址逻辑复杂。同时,较多未完成累加的中间结果暂存也给加速器缓存资源带来极大负担。因此需要在利用全对全数据流加速推理计算的基础上,设计一种数据加载方案,以简化输出数据的累加逻辑,并降低输出数据寻址存储的复杂度。该方案主要分为两部分:①针对数据加载顺序,通过将 IFM 数据和权重按照一定先后次序加载到乘加阵列,简化输出数据的中间结果累加逻辑,有利于通过一定规律将输出数据寻址暂存;②针对数据复用,通过减少 IFM 数据从片外存储单元读取的次数以及片内暂存的中间结果数量,实现数据搬移功耗和输出缓存需求的减少。

2.2.1　数据加载预处理

加载数据前,为了剔除 0 值稀疏数据并减少存储空间需求,对 IFM 数据和权重均采用压缩稀疏行(compressed sparse row, CSR)编码方式存储,并在计算过程中实时解码恢复其原始坐标。通过 CSR 编码,将得到由非 0 数据组成的密集数据向量,以及由非 0 数据间 0 值数据的数量组成的数据偏移量。如图 5 所示,IFM 数据在进入输入缓存单元前,首先由稀疏解码模块根据 IFM 数据偏移量还原出 IFM 数据的行、列坐标,然后 IFM 数据及其行、列坐标组成输入数据加载至数据分配单元(demultiplexer, DMUX),最后将不同列坐标的 IFM 数据经数据分配单元分配到不同编号的 IFM FIFO(first in first out)中,IFM FIFO 的数量依乘加阵列的行数 N 而定。指针控制模块用于控制多个 IFM FIFO 并行输出。

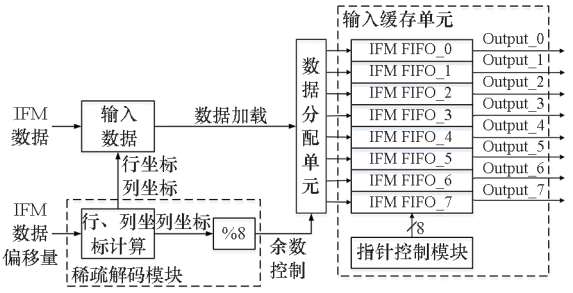


图 5　IFM 数据加载至输入缓存单元

Fig. 5　Loading the IFM data into the input buffer unit

图 6 展示了稀疏输入数据存入输入缓存单元的方式,左图稀疏输入数据的部分空缺位置是 IFM 上未被压缩存储的 0 值数据,由于 FIFO 的特性,后续数据会顶替其位置,保证了 FIFO 数据的密集性,使得每个计算周期每个 FIFO 都能输送数据给乘加阵列处理,有效充分利用乘法器资源,配合全对全数据流计算完成更高效的 DNN 推理计算。同时从图 6 中可以看到,若稀疏随机度较高,则可能会在不同的 FIFO 中产生非等长的队列,进而导致在某些周期中乘法器未得到充分利用。这一问题可通过均衡稀疏[14-16]方法减轻其带来的影响。

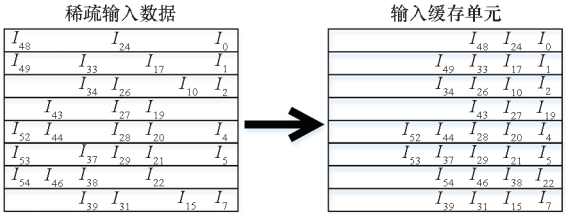


图 6　稀疏输入数据存入 FIFO

Fig. 6　Sparse input data storage into FIFO

如图 7 所示,稀疏编码存储的权重矩阵以相同的方式进行稀疏解码后送入 8 路 FIFO 中缓存,FIFO 数量依乘加阵列的列数 M 而定。

当权重缓存单元输出权重时,8 路 FIFO 可以输出不同输出通道的卷积核权重到乘加阵列的不同列,便于并行执行不同输出通道的推理计算。

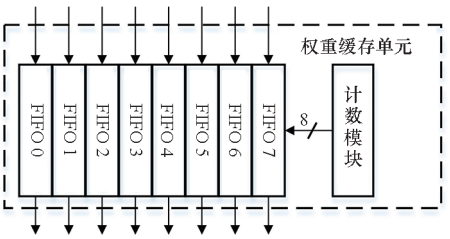


图 7　权重缓存单元结构

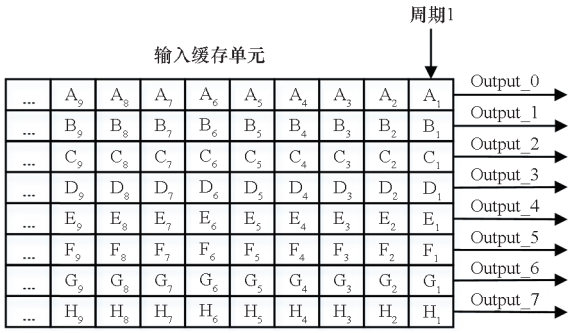
Fig. 7　Weight buffer unit structure

计数模块则用于控制 FIFO 的权重读取。

2.2.2　输入驻留

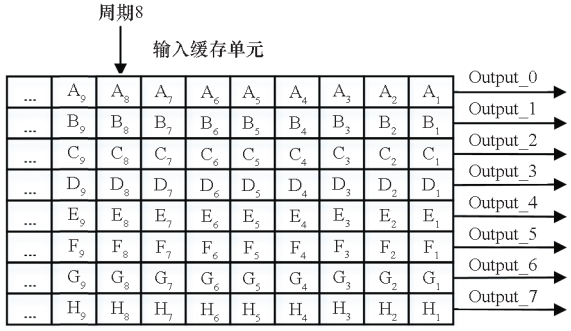
在进行 DNN 数据处理时，每次需向输入缓存单元加载具有相同输入通道编号的 IFM 数据和所需的全部权重，提高所加载数据在推理计算过程中的复用率可以极大程度减少数据的反复提取。相较其他数据复用方式，数据驻留可以最小化读取数据时所带来的能耗[17]。

本设计采用输入驻留的数据复用方式，将已加载的 IFM 数据驻留在输入缓存单元中，待相同输入通道的 IFM 数据与全部权重完成计算后，更换下一个 IFM 数据，以此最小化 IFM 数据的读取和写入能耗。如果输出 IFM 数据时单纯地保持其完全不变，就需要持续更新权重，会导致权重缓存单元面临频繁的数据交互。为了避免这种情况，本设计选择一定周期内维持权重不变，如图 8 所示，首先，输入缓存单元的每个 FIFO 以 k 个 IFM 数据为一组进行周期性循环输出，每 k 个周期为一次循环，每个周期输出该组的不同 IFM 数据；其次，同一个循环内，权重缓存单元并行输出 1 个权重保持 k 个周期不变；然后，下一次循环从头开始输出该组 IFM 数据并更换下一个权重；最后，当该组 IFM 数据遍历全部权重后，更新下一组 IFM 数据开始周期性循环输出。在循环期间，IFM 数据组一直驻留在 FIFO 内，由指针控制模块控制 FIFO 读取指针的变化，以此实现输入驻留。维持权重不变的工作则由计数模块完成，通过计数 k 个周期读取一次权重，保证每个权重在加速器内均复用 k 次。由于输入数据需要根据列坐标被输送到不同的 FIFO，每个周期只能完成 1 个输入数据分配，故 8 个 IFM FIFO 并行输出 IFM 数据组时数据量 k 不宜过大，否则会导致输入数据分配时序紧张。而且 IFM 尺寸随着网络深度的增加而减小，另外存在稀疏性，每列的 IFM 数据也较少。所以本设计设置 k 为 8，在缓解输入数据分配压力的同时尽可能多地复用权重。具体表现为：图 8(a)表示第 1 个周期输出下标为 1 的 IMF



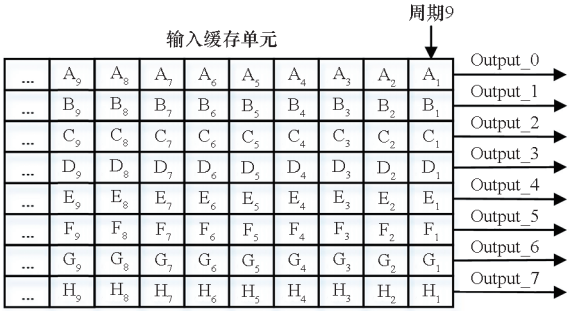
(a) 第 1 周期 IFM 数据输出

(a) IFM data output in cycle 1



(b) 第 8 周期 IFM 数据输出

(b) IFM data output in cycle 8



(c) 第 9 周期 IFM 数据输出

(c) IFM data output in cycle 9

周期	1	2	3	4	5	6	7	8	9
FIFO_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_1
FIFO_1	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_1
FIFO_2	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_1
FIFO_3	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_1
FIFO_4	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_1
FIFO_5	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_1
FIFO_6	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_1
FIFO_7	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8	H_1
权重	W_1								W_2

(d) 不同周期的数据输出

(d) Data output for different cycles

图 8　输入驻留

Fig. 8　Input stationary

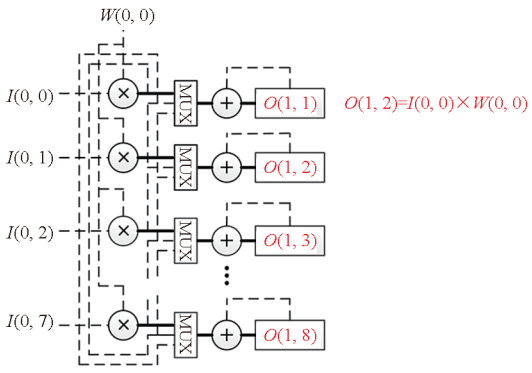
数据;图 8(b)表示第 8 个周期输出下标为 8 的 IFM 数据;图 8(c)表示第 9 个周期重新输出下标为 1 的 IFM 数据,以此循环往复;图 8(d)表示在连续的 8 个周期内,只有 IFM 数据变化,而权重不变,用 W_1 替代权重缓存单元并行输出的权重。

2.3　输出数据高效缓存

全对全数据流计算用于完成 DNN 推理计算加速;输入驻留用于高效加载数据至乘加阵列,减少数据搬移。本小节所介绍的输出缓存分块则用于解决乘加阵列计算结果的累加、暂存等问题。

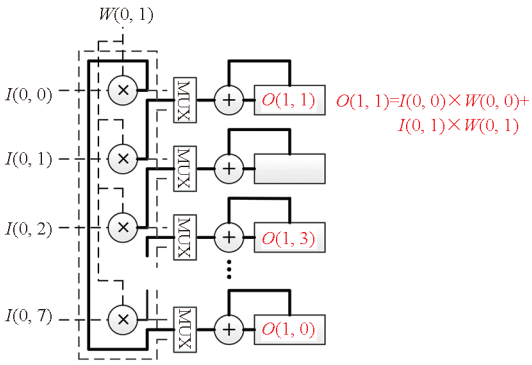
通过全对全数据流计算得到的乘积结果需要被有效存储,并能方便后续输出数据中间结果的累加。如果试图将乘加阵列每个计算周期得到的 64 个计算结果输送至一个输出缓存单元中暂存,受限于输出缓存单元每周期只能存储有限数据个数,就会导致存储的时序无法满足大量数据缓存需求,引发数据拥塞。因此,本设计将输出缓存单元划分成多块,为每个乘法器搭配独立的缓存模块,以保证每个乘积结果能立即存储,避免时序紧张。

为简化计算结果缓存寻址难度,本设计首先限制乘加阵列中不同列的权重来自不同输出通道的卷积核,因此乘法器列与列之间的计算可以互不影响。以单列乘法器输出结果的数据流传输为例介绍所设计的输出缓存架构:如图 9(a)所示,在密集 DNN 计算中,8 列 IFM 数据分别加载到不同行的乘法器,与坐标为(0,0)的权重进行乘积,得到对应输出数据的第一个中间结果,无须累加可直接存入输出缓存单元。图 9(b)中更换坐标为(0,1)的权重时,此时 8 列 IFM 数据与权重的乘积结果需要错位一行传输,与上一行输出数据的第一个中间结果累加,亦即,此时第二行的乘积 $I(0,1)\times W(0,1)$需与第一行存储结果 $I(0,0)\times W(0,0)$累加。由于 $I(0,0)\times W(0,0)$存储在第一行的输出缓存单元中,故 $I(0,1)\times W(0,1)$需向上传输至第一行完成累加后并存储在第一行,第三行的乘积结果传输至第二行,以此类推。图 9(c)中更换坐标为(0,2)的权重时,此时 8 列 IFM 数据与权重的乘积结果需错位两行传输,同理,第三行的乘积结果 $I(0,2)\times W(0,2)$需要与第一行存储的中间结果 $I(0,0)\times W(0,0)+I(0,1)\times W(0,1)$累加,故 $I(0,2)\times W(0,2)$需传输至第一行完成累加后并存储在第一行,第四行的乘积结果传输至第二行,以此类推。根据权重列坐标的不同,当前行的乘积结果需向上传输至不同行。而更改为不同行的权重时,需要累加的 IFM 数据也会更改,但列坐标相同的 IFM 数据会加载至同一行的乘法器,即 $I(0,0)\times W(0,0)$与 $I(1,0)\times W(1,0)$均在第一行乘法器完成乘积,故无须错行传输。



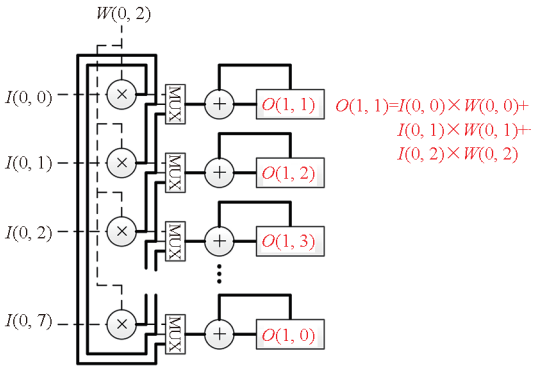
(a) 中间结果直接存储

(a) Intermediate result storage directly



(b) 中间结果向上传输一行并累加存储

(b) Intermediate result transfers up one line to accumulate and store



(c) 中间结果向上传输两行并累加存储

(c) Intermediate result transfers up two rows to accumulate and store

图 9　单列乘法器输出结果的数据流传输

Fig. 9　Data flow transmission of single column multiplier output results

综上所述,连续三行乘法器输出需汇集到同一行多路选择器(multiplexer, MUX),由权重列坐标控制数据输出并完成累加,累加结果存储在当前行

的输出缓存单元,实现了单周期内完成输出数据的累加和暂存,以此解决了数据拥塞的问题;同时,不同列的乘加运算互不影响,不同行的数据存储相互独立,以此解决了输出数据访问争用的问题。

3 实验结果与分析

基于 Verilog HDL 语言对设计完成了单个 PE 的寄存器传输级(register transfer level, RTL)实现,利用 Synopsys Design Compiler 工具基于 FreePDK 45 nm 工艺进行了综合,并完成对设计的性能分析:面积评估、数据加载效率仿真、乘法器利用率分析。PE 架构与 FSA 有较多相近之处,如:①二者都采用了乘法器阵列,且阵列中每行(列)均共享相同的 IFM 数据(权重);②都在进行卷积运算的同时计算 OFM 数据的存储坐标;③IFM数据和权重加载前均无须进行配对等。同时,相比其他支持稀疏计算的加速器, FSA 在性能和能效等方面均展示了较明显的优势,其中在性能方面与 SCNN、CCR、Cambricon-S[3] 相比分别提高了 173%、123%、108%,在能效方面分别提高了 1 386%、924%、625%,因此将 FSA 作为实验的比较基准。

3.1 面积评估

基于稀疏卷积计算高效数据加载和输出缓存策略设计了图 10 所示的处理单元结构。IFM 数据和权重偏移量经过稀疏解码单元得到对应的行、列坐标。IFM 数据和权重根据这些坐标经数据选择单元输入至输入缓存单元和权重缓存单元。MAU 主要由乘法器、选择器、地址计算单元、加法器、输出缓存单元构成。输出缓存单元可根据不同行数据分为 4 个区域,每个区域存储多个输出通道的数据。

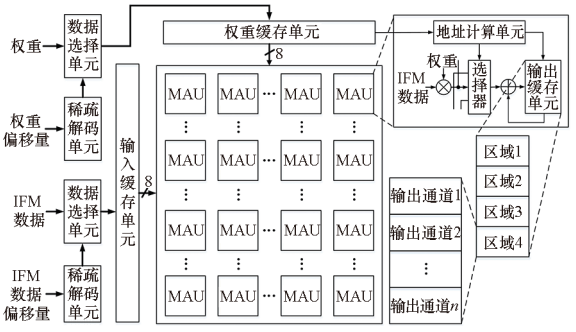


图 10 处理单元顶层结构

Fig. 10 Processing element top-level structure

由于处理单元在获取 IFM 数据和权重时,每次可最多获取连续的 64 个数据,考虑到最坏的情况,即这 64 个数据均被分配到同一个 FIFO,因此 FIFO 深度至少需要设置为 64。

输入缓存单元中,每个 FIFO 不仅要存储8 bit 的 IFM 数据,还需要将其行、列坐标进行存储。以 VGG-16 为例,最大行、列坐标为 224,则需要用 8 bit 表示行、列坐标信息,因此输入缓存单元实际存储空间大小为:

$$\text{输入缓存空间} = 64 \times 8 \times (8+8+8)\ \text{bit} = 12\ \text{Kbit} \tag{4}$$

因此,输入缓存单元存储空间大小为 1.5 KB。同理,权重缓存单元也按照需求设置为 1.5 KB。

对于输出缓存单元,处理单元能存储 4 行完整的 OFM 数据,在得到完整数据后,输出缓存单元便输出数据以进行池化、激活等操作。故输出缓存空间的大小由实际运行的网络决定,以 VGG-16 为例,第一层卷积的输出特征图像尺寸为 224 像素 ×224 像素 ×64 像素,当输出数据位宽为 24 bit 时,输出缓存空间为:

$$\text{输出缓存空间} = 4 \times 224 \times 64 \times 24\ \text{bit} = 1\ 344\ \text{Kbit} \tag{5}$$

因此,输出缓存单元存储空间大小为 168 KB。

基于 FreePDK 45 nm 工艺对设计完成 RTL 到电路的实现,并根据工艺尺寸比的平方等比例缩小,将设计参数等效为 28 nm 工艺下的参数并与 FSA 进行比较。表 1 展示了本设计与 FSA 在单 PE 上的参数对比,从整体上看,本设计处理单元在设计需求上主要集中在缓存单元,但针对不同神经网络应用,所需存储的输出数据量各不相同,故在实际应用中允许对输出缓存单元进行可配置性设计。由于简化了输出缓存的逻辑设计,故不需要对累加缓存单元和仲裁交叉开关进行设计,PE 总体面积相比 FSA 减少了 21.45%。

表 1 本文策略与 FSA 在 PE 上的参数比较

Tab. 1 Parameters comparison between this paper and FSA on PE

PE 参数	本文策略	FSA
工艺	FreePDK 45 nm 等效 28 nm	28 nm
缓存空间	171 KB	30 KB
缓存面积	0.416 mm^2	0.073 mm^2
乘法器阵列	0.033 mm^2	0.033 mm^2
累加缓存单元		0.393 mm^2
仲裁交叉开关		0.115 mm^2
其他单元	0.126 mm^2	0.118 mm^2
PE 总计面积	0.575 mm^2	0.732 mm^2

3.2　数据加载效率仿真

数据加载效率能反映出 PE 的处理性能。在进行 DNN 推理计算时,本设计所需的数据加载周期与输入缓存单元内数据量最大值以及权重缓存单元内数据量最大值有关。这是因为:受数据稀疏的影响,在不考虑 FIFO 深度的情况下,将单输入通道的 IFM 分配到输入缓存单元时,8 个 IFM FIFO 获取的数据量并不相同,而同一输入通道的数据没有全部完成处理就无法开始下一个通道的数据处理,所以数据加载周期与输入缓存单元内 FIFO 的最大数据量有关,令 8 个 FIFO 中的数据量最大值为 Max_{I}。同理,令权重缓存单元的数据量最大值为 Max_{W}。每个 IFM 数据需要与权重缓存单元中的所有权重进行乘积,则每个 IFM 数据需要循环 Max_{W} 次,单输入通道的 IFM 数据加载周期 $Cycle_{\mathrm{channel}}$ 表示为:

$$Cycle_{\mathrm{channel}} = Max_{\mathrm{I}} \times Max_{\mathrm{W}} \tag{6}$$

完成一个输入通道的数据加载后开始其余输入通道的数据加载,则单个卷积层的数据加载周期 $Cycle_{\mathrm{layer}}$ 是所有输入通道的周期之和:

$$\begin{aligned} Cycle_{\mathrm{layer}} &= \sum_{i=1}^{C_i} Cycle_{\mathrm{channel}}^{i} \\ &= \sum_{i=1}^{C_i} (Max_{\mathrm{I}}^{i} \times Max_{\mathrm{W}}^{i}) \end{aligned} \tag{7}$$

其中,C_i 表示该层 IFM 的输入通道数。一幅图像所需的周期是所有卷积层的周期之和:

$$Cycle_{\mathrm{total}} = \sum_{i=1}^{Layer} Cycle_{\mathrm{layer}}^{i} \tag{8}$$

与 FSA 相比,其 PE 内的数据处理是将 IFM 数据根据乘法器阵列大小分成多组,每个周期依次输入不同组 IFM 数据并加载不同的权重。如图 11 所示,第 1 个周期加载第 1 组 IFM 数据和第 1 个权重到乘法器阵列的第 1 列;第 2 个周期将第 1 列的 IFM 数据水平移位到第 2 列,并加载第 2 组 IFM 数据到第 1 列;第 1 个权重固定在第 1 列不动,加载第 2 个权重至乘法器阵列的第 2 列。根据此数据流,对 FSA 以及本设计进行数据加载效率仿真,分析二者处理相同神经网络以及网络中不同层的数据加载所需周期。

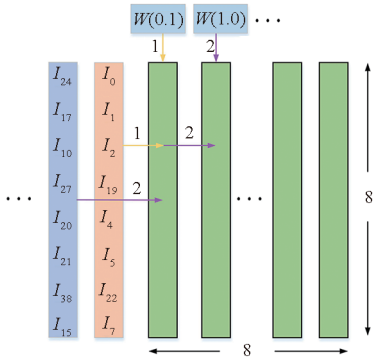


图 11　FSA PE 数据流

Fig. 11　FSA PE data flow

在对比数据加载效率前,对不同网络层的数据稀疏度进行设定。SCNN[12] 展示了 VGG – 16 不同层 IFM 和权重的稀疏度,整理数据如表 2 所示。

表 2　VGG – 16 不同卷积层 IFM 和权重的稀疏度[12]

Tab. 2　Sparsity of IFM and weight for different convolutional layers in VGG – 16[12]

%

卷积层	IFM 稀疏度	权重稀疏度
Conv1_1	0	42
Conv1_2	50	79
Conv2_1	12	65
Conv2_2	21	64
Conv3_1	19	44
Conv3_2	37	77
Conv3_3	30	58
Conv4_1	28	67
Conv4_2	49	74
Conv4_3	54	64
Conv5_1	61	63
Conv5_2	64	70
Conv5_3	68	63

根据表 2 设置不同层的数据稀疏度对本设计以及 FSA 处理 VGG – 16 时的数据加载周期进行仿真验证,图 12 展示了二者在处理 VGG – 16 时不同卷积层的数据加载周期对比。对于每一个卷积层,本设计所需的数据加载周期均小于 FSA。从总的数据加载周期来看,本设计所需时间仅为 FSA 的 45.93%。由于数据加载至乘加阵列能立即执行计算,故所需数据处理周期与数据加载周期一致,因此在数据处理速度方面提升了 117.71%。以上数据对比在权重随机分布的情况下实现,即权重在卷积核的分布完全随机且无规律,实验结果验证了本设计在任何情况都比 FSA 更加高效。

事实上,权重分布可以通过训练和剪枝改变,并根据处理需求进行适当调整,比如 Yao 等[14] 提出了一种均衡稀疏和对应剪枝算法,可以均衡不

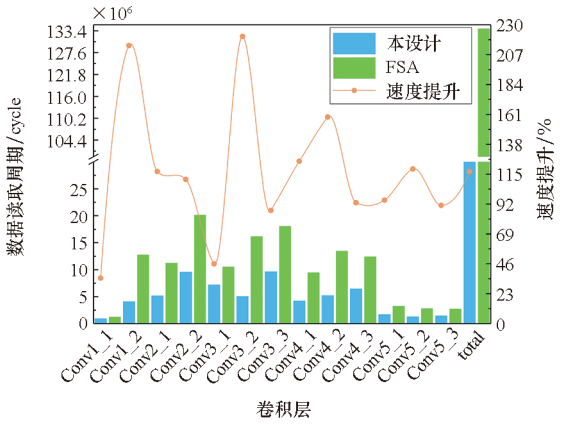


图 12　本设计与 FSA 处理 VGG－16 时不同卷积层的数据加载周期对比

Fig. 12　Comparison of data loading cycles for different convolutional layers with VGG－16 processing in this design and FSA

同卷积核的权重数量，在保持高计算精度的同时能够实现卷积核并行处理加速。在完成卷积核训练后，权重分布基本维持不变，不同卷积核的权重数量也实现均衡，使得每个权重缓存单元的权重数量维持在平均值左右。在此基础上，稀疏 DNN 推理计算时所需的数据加载周期会获得减少，从而提高推理计算速度。如图 13 所示，本设计实现权重均衡后数据加载周期与之前相比，总体减少了约 11.87%。

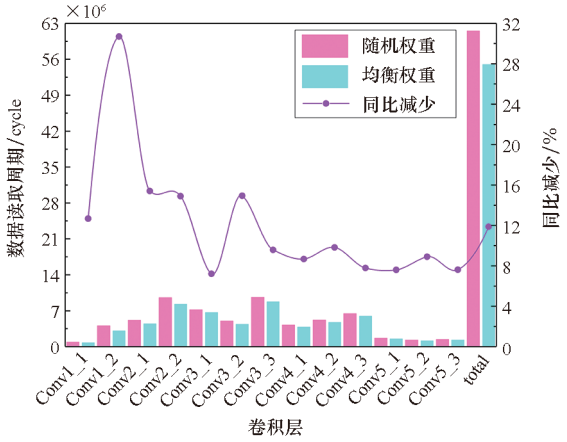


图 13　本设计在均衡权重与随机权重的数据读取周期对比

Fig. 13　Data read cycles of this design in balanced weights and random weights

3.3　乘法器利用率分析

乘法器的利用率反映出所设计电路实际的资源使用情况，优秀的设计应充分利用资源，实现计算量的最大化以及推理计算加速。根据式(1)，乘法器利用率是有效计算量与计算总量之比，在电路系统中，应是有效计算量与乘法器可使用量之比，如式(9)所示：

$$\text{乘法器利用率}=\frac{\text{计算总量}-\text{冗余计算量}}{\text{数据加载周期}\times 64} \tag{9}$$

每个数据加载周期都有一定数量的乘法器供计算使用，处理单元每周期可使用 64 个乘法器同时计算，而充分利用资源是实现计算加速的途径之一。图 14 展示了在均衡权重的情况下，本设计与 FSA 单个 PE 在不同卷积层的乘法器利用率。在层数较浅时，得益于权重均衡，乘法器利用率能维持在 95% 以上；在层数较深时，由于 IFM 尺寸的降低，每个通道的待处理数据过少而引起乘法器利用率下降。但总的来看，平均乘法器利用率达到 89%。相比之下，FSA 的乘法器利用率并不乐观，但是 FSA 设计了一个混合分区方案，即层数较浅时，将 IFM 按尺寸分成多块；层数较深时，将 IFM 按通道分成多块，每个 PE 同时处理不同块的数据。其通过混合分区可将平均乘法器利用率提高到 80%，但本设计仍比 FSA 的乘法器利用率提高了 11.25%。

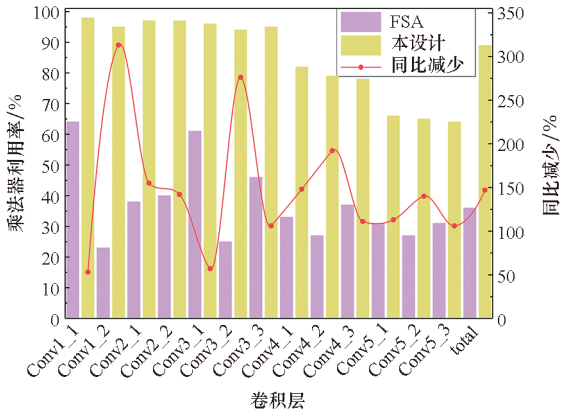


图 14　均衡权重下本设计与 FSA 的乘法器利用率对比

Fig. 14　Comparison of multiplier utilization between this design and FSA under balanced weights

4　结论

本文提出了稀疏卷积计算高效数据加载与输出缓存策略，通过采用全对全数据流计算，即将属于同一输入通道的非零 IFM 数据与非零权重进行任意两两配对计算，降低了非零数据配对难度，提高了乘加资源利用率；其次通过采用输入驻留计算以及密集型循环加载 IFM 数据，大幅减少了数据片外调取次数；最后优化了输出缓存设计并简化了乘累加中间结果缓存寻址过程，实现了输出数据的简便、快速存储，避免了数据访问争用、存储拥塞等问题。实验表明，与采用类似架构的

FSA 相比,在处理单元面积上减少了 21.45%;在数据加载速度方面平均提高了 117.71%;在平均乘法器资源利用率方面提高了 11.25%,并达到 89%。

参考文献(References)

[1] WEN J Y, MA Y F, WANG Z F. An efficient FPGA accelerator optimized for high throughput sparse CNN inference[C]//Proceedings of IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 2020: 165 - 168.

[2] ZHANG H, GOU A R, FAN Y B, et al. A fine-grained sparse neural network accelerator for image classification [C]// Proceedings of IEEE 14th International Conference on ASIC (ASICON), 2021: 1 - 4.

[3] ZHOU X D, DU Z D, GUO Q, et al. Cambricon-S: addressing irregularity in sparse neural networks through a cooperative software/hardware approach[C]//Proceedings of 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2018: 15 - 28.

[4] CHEN Y J, LUO T, LIU S L, et al. DaDianNao: a machine-learning supercomputer [C]//Proceedings of 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2015: 609 - 622.

[5] ALBERICIO J, JUDD P, HETHERINGTON T, et al. Cnvlutin: ineffectual-neuron-free deep neural network computing [C]//Proceedings of ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016: 1 - 13.

[6] ZHANG S J, DU Z D, ZHANG L, et al. Cambricon-X: an accelerator for sparse neural networks [C]//Proceedings of 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016: 1 - 12.

[7] LI J J, YAN G H, LU W Y, et al. CCR: a concise convolution rule for sparse neural network accelerators[C]// Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018: 189 - 194.

[8] LU L Q, XIE J M, HUANG R R, et al. An efficient hardware accelerator for sparse convolutional neural networks on FPGAs [C]//Proceedings of IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2019: 17 - 25.

[9] LIU B S, CHEN X M, HAN Y H, et al. Search-free accelerator for sparse convolutional neural networks [C]// Proceedings of 25th Asia and South Pacific Design Automation Conference (ASP-DAC), 2020: 524 - 529.

[10] LIN C Y, LAI B C. Supporting compressed-sparse activations and weights on SIMD-like accelerator for sparse convolutional neural networks [C]//Proceedings of 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), 2018: 105 - 110.

[11] CHANG K W, CHANG T S. VSCNN: convolution neural network accelerator with vector sparsity[C]//Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS), 2019: 1 - 5.

[12] PARASHAR A, RHU M, MUKKARA A, et al. SCNN: an accelerator for compressed-sparse convolutional neural networks[C]//Proceedings of the 44th Annual International Symposium on Computer Architecture, 2017: 27 - 40.

[13] LI F R, LI G, MO Z T, et al. FSA: a fine-grained systolic accelerator for sparse CNNs [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(11): 3589 - 3600.

[14] YAO Z L, CAO S J, XIAO W C, et al. Balanced sparsity for efficient DNN inference on GPU [C]//Proceedings of the AAAI Conference on Artificial Intelligence, 2019, 33: 5676 - 5683.

[15] WU D, FAN X T, CAO W, et al. SWM: a high-performance sparse-winograd matrix multiplication CNN accelerator [J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2021, 29(5): 936 - 949.

[16] XIE X R, LIN J, WANG Z F, et al. An efficient and flexible accelerator design for sparse convolutional neural networks[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2021, 68(7): 2936 - 2949.

[17] SZE V, CHEN Y H, YANG T J, et al. Efficient processing of deep neural networks: a tutorial and survey [J]. Proceedings of the IEEE, 2017, 105(12): 2295 - 2329.