算法 & 架构整理 v1.0

Fan Yinghui

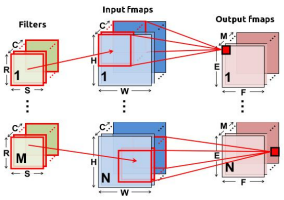
January 29, 2018

1 硬件架构

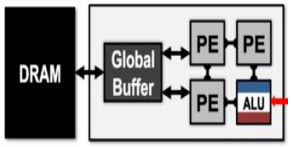
图一给出了卷积 (Convolution) 计算的伪代码、示意图和典型的加速器架构图。从伪代码中可以看出卷积计算有 6 个循环，基本计算为乘加计算，循环体中计算的索引小标则反映了循环间的数据复用和数据依赖。典型的加速器架构包括 global buffer(一般几百 KB，与 DRAM 相连)、PE 阵列 (PE 之间有一定互联关系)、每个 PE 中还有 Registor File(RF 一般几 KB 或更小)。

```
for(m=0;m<M;m++){      //output feature maps
 for(n=0;n<N;n++){     //input feature maps
  for(r=0;r<S;r++){    //output neuron rows in a feature map
   for(c=0;c<S;c++){   //output neuron columns in a feature map
    for(i=0;i<K;i++){  //kernel rows operations
     for(j=0;j<K;j++){//kernel columns operations
     L:                //loop body (multiplication-accumulation)
       O(m)(r,c) += K(m,n)(i,j) x I(n)(r+i,c+j) ;
} } } } } }
```

(a) 卷积伪代码



(b) 卷积图示



(c) 加速器图示

Figure 1: 卷积计算及硬件架构

加速器设计首先要考虑的问题即设计合适的 dataflow 将卷积计算映射到 PE 阵列和存储层次中，有两个考虑的角度，一是充分利用计算中的数据复用，减少可复用数据的访存次数，由此降低功耗; 二是充分利用计算中的可并行性，选择合适的循环展开方法，增大吞吐量。

1.1 数据复用

加速器的存储结构包括 PE 单元内部的 RF、片内的 Global Buffer 以及片外 DRAM，从不同的存储单元中读取数据所消耗的能量不同。而卷积的计算中提供了三种数据复用的可能：1. convolution reuse; 2. featuremap reuse; 3.filter reuse。

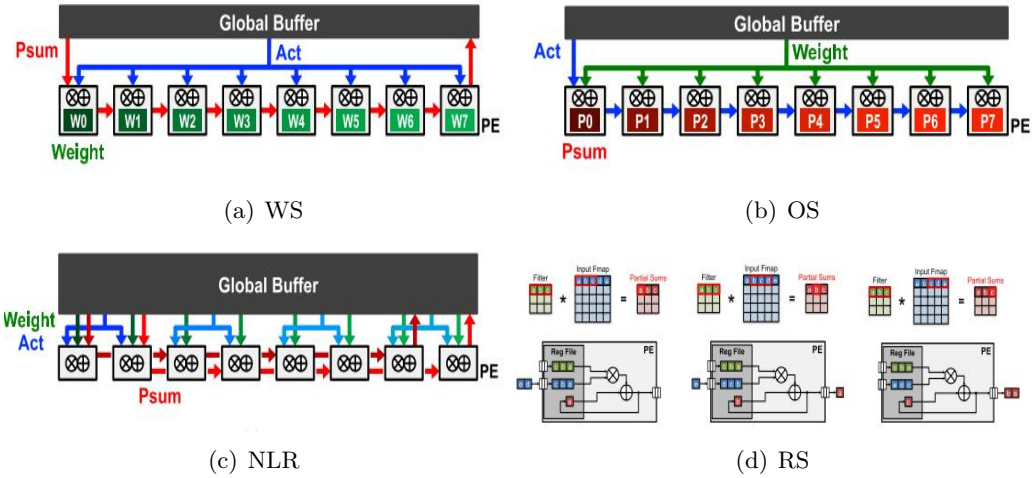
因此从数据复用角度看，尽可能将可复用的数据存在 local Memory 中，减少从 DRAM 中读取的次数，则可以降低功耗。根据将何种数据存在 local mem 中，dataflow 可以分为以下几种 [11]:

1.1.1 Weight Stationary

将不同的权重存在 RF 中，并尽可能多的利用这些权重进行计算。因此每个 PE 的 RF 存储不同的权重，同一激活层数据广播至所有的 PE 单元，而同一个 Psum 通过每一个 PE 单元并累加获得最终结果。代表案例是 NeuFlow[5]。

1.1.2 Output Stationary

将中间和 Patial Sum 存在 RF 中，减少从 buffer 和 DRAM 中读取 Psum 的次数。因此 Psum 存在 RF 中不动，将同一权重广播至所有 PE 单元，而同一个 Act 依次通过每一个 PE 单元。Psum 可以来自同一通道不同位置、不同通道同一位置或不同通道不同位置。代表案例是 ShiDianNao[4]。



(a) WS (b) OS

(c) NLR (d) RS

Figure 2: Dataflow

1.1.3 No Local Reuse

不再使用 local 的 RF，全部使用 global buffer，减少与 DRAM 的数据交换。因此通过复制多份 Act，与权重计算，并通过累加各个 PE 的 Psum 获得最终结果。代表案例是 DianNao[1] 和 DaDianNao[3]。

1.1.4 Row Stationary

针对所有类型的数据 (w,act ，psum) 最大化 RF Level 的数据复用，其计算方法如 Figure2(d)。代表案例是 Eyeriss[2]。

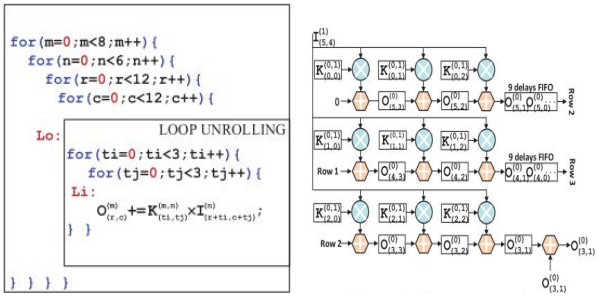
1.2 循环展开

加速器的设计中一般通过将循环展开并行化的方式来实现加速，卷积计算共有 6 个循环可以展开，根据循环展开的策略，卷积计算中的并行度分为 3 类，Feature

map Parallelism(FP, 对应循环 m,n)，Neuron Parallelism (NP, 对应循环 r,c),Synapse Parallelism (SP，对应循环 i,j)，考虑是否支持这 3 种并行度，共有 8 种相应的 dataflow。目前大部分加速器设计是其中的三种 dataflow，对应的架构依次是 Systolic、Tile-based 和 2D-Mappling。

1.2.1 SFSNMS

按 kernel 的 index 展开，SFSNMS，每次取一个 input，broadcast 至所有的 PE，每个 PE 取一个权重 (3x3 kernel)，计算结果为 9 个对应位置的部分和。每个 cycle 部分和向下一个 PE 移动或进入队列或下一行，完成后最终输出 output 某一位置的结果。数据复用：相邻位置的 output 被 mapping 到不用的 PE 中，空间上复用了同一个输入，时间上通过每个 PE 的寄存器复用了权重数据。代表架构 (systolic):Neuflow[5]。

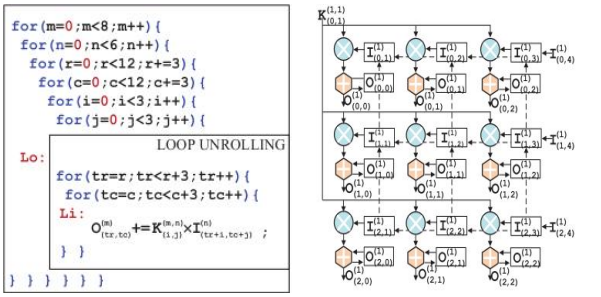


(a) Loop Unroll (b) Dataflow

Figure 3: SFSNMS

1.2.2 SFMNSS

按照 output 的位置索引展开，一次计算多个 output 位置的结果。然后输入多个位置的 input 和同一个权重，输出多个位置 output。从右向左从上向下传递累加，最终输出。数据复用：相邻位置的 output 被 mapping 到不同的 PE 中，空间上复用了同一个权重，通过 PE 的连接和寄存器时间上复用了 input。代表架构 (2D-Mapping):ShiDianNao[**?**]。

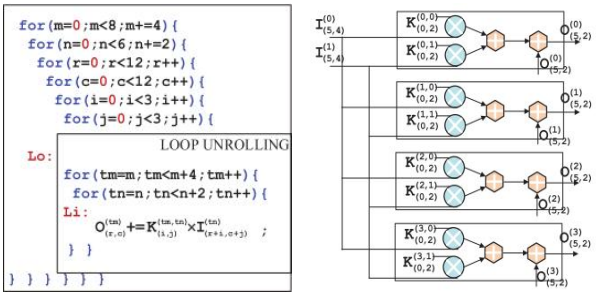


(a) Loop Unroll (b) Dataflow

Figure 4: SFMNSS

1.2.3 MFSNSS

按照通道索引展开，输入同一位置不同通道的 input 和不同通道不同 kernel(对应 output 的维度) 的权重，输出同一位置不同通道的 output。数据复用：不同的 PE 空间上复用了同一个 input，但无法在空间上复用权重和 output，这种计算架构数据复用最小。代表架构 (tiling):DianNao[?]、DaDianNao[3]。



(a) Loop Unroll (b) Dataflow

Figure 5: MFSNSS

1.3 总结分析

数据复用与循环展开是对卷积计算分析的两个角度，两者之间存在着一定的对应关系。数据复用可以看作将 input、output 或 weight 之一的 index 展开，反映在硬件上即是这些数据存在 local mem 中不动，可调整的 index 则体现了循环展开的可能性。以 Neoflow 为例，按照数据复用分类其属于权重复用，每个 PE 存一个不同的权重并尽可能多地利用这个权重进行计算，权重的不同体现在索引 {i,j} 的不同，对应循环展开即是 SFSNMS。

实际加速器设计中，dataflow 可以是这些分类的组合或变种，其形式更为灵活 [8]; 此外, dataflow 与硬件架构也不是严格的一一对应, 例如 Diannao 和 kernel partitia[10]l。

这两个分类方法为分析加速器的 Dataflow 和架构设计提供了非常好的参考，一般的设计其数据复用和循环展开都会落在这样的分类下。不过能否从分类正向推导或排列组合出所有可能的形式？要考虑的问题太多还想不清楚。

1.4 与 Aladdin 对比

Aladdin 建模的硬件对象主要是存储和控制部分，不包括控制逻辑。其中存储包括 ScrarchPad(buffer) 和 Register，可调整的参数为存储资源大小、数据排布方式、带宽、位宽等。计算部分则是并行的 PE 阵列 (Datapath) 组成，可调整的设计参数为并行度和流水线。

Aladdin 的 loop unrolling 体现了负载应用在硬件中并行执行的情况，但无法体现这些并行单元之间的信息交互。1.1 中的数据复用是指在 RF 这个层次的复用，要求数据能在不同的 PE 间传输，Aladdin 的目前建模方法显然无法满足，只能支持 no local reuse 这一类型，即不需要 PE 内的 RF，其数据复用体现在 global buffer 这个层次。而从循环展开的角度看也是这样，Aladdin 要求展开循环的计算不能有 PE 间交互的需求。

综上所述，Aladdin 目前所能建模的硬件架构典型结构是 Tile-based，可行的循环展开方式有?。

2 量化算法

2.1 算法分类

考虑 CNN 量化后对硬件设计的影响，将量化算法分为两类：一是近改变计算位宽不需要改变计算单元类型，代表案例是 Dorefa-Net; 二是需要新的计算单元设计，例如 BNN、logNet 等。

2.2 实验分析与数据统计

算法部分以 Cifar10 (cifar100) 为测试数据集，参考 AlexNet 设计相应的网络架构，需要注意：Baseline 精度是否足够、网络结构复杂度是否足够 (参考硬件设计中测试用的网络结构)。在数据集和网络架构固定的情况下，进行不同量化算法的对比实验，除了统计模型精度外，还应计算的参数如下：

Table 1: 统计数据

	精度	MAC 总数	Mem 占用
网络层			
量化算法			
位宽			

Table 2: 网络架构参数

	XXNet	
Input Size		
Num of Conv Layers		
Num of FC Layers		
Num of Channels		
Num of Filters		

3 针对量化的计算单元和存储调整

量化算法应用于卷积神经网络后，数值精度改变进而改变表示数据所需的位宽，反映在硬件上主要包扩计算和存储两大类问题，即数据表示所需的位宽与硬件计算、存储的位宽之间不匹配，下面分别讨论。

3.1 计算

1) one-size-fits-all：在数据位宽小于等于计算位宽情况下，用相同的最大所需配置执行所有计算，代表案例 DianNao[1]。

2) operand narrowing：窄位宽的数据通过宽的计算单元时，低位为 0, 晶体管开关活性降低，以此降低功耗。这样设计的好处是复杂度低，不需要做太多适配与调整，但存在资源的浪费。代表案例 envision[9]。

3) operator narrowing：另外一种情况是宽数据通过窄计算单元，如下图 c 所示。通过时间上复用来实现 wide 操作数的计算，但增加了 latency。此外，有一些 overhead 例如额外的寄存器处理分时复用的中间存储，适合用在 SIMD 中增加并行度。代表案例 stripe[7]。

4) special case：在某些特殊量化情况下，原有的乘加计算会退化为更简单的逻辑，于是可以使用更简单的计算单元。代表案例 BNN[13]，Shifter-based[12]。

3.2 访存

1) Fixed-length：当数据位宽发生改变，依然可以用统一的存储位宽，低位置零不使用即可，设计复杂度低，但不能有效节省存储空间。

2) Data Packing and Unpacking。将不同数据宽度的 data 打包到 fixed-length 的存储空间中，减少了带宽和存储的需求，但增加了访问数据的复杂度。这里也分两个层次，当数位宽度是 2 的指数时，恰好可以 pack 在 32bit 存储里; 而当数位宽度更灵活，如 7,9 等，即便 pack 在 32bit 数里，也存在数位对齐的问题。代表案例 proteus[6]。

4 总结

结合总结，将涉及到的从算法到硬件的设计空间列表整理如下。

Table 3: 设计空间汇总

Level	Space	..	Implementation
Arch	Tile-Based	Tm,Tn,Buffer Size, MemPorts	Configure
Data flow	MFSNSS		C code
Quantization	Float Fixed point Power of 2 Binary	32bit 2,4,8,16 bit 2,4,8,16 bit 1 bit	Pytroch
Compute Unit	float mult fixed mult Shifter Xnor	与算法对应 bit-parallel bit-serial	PPA model based on RTL
Mem Access	Pack & Unpack	Buffer Size, MemPorts	添加 Node ,C Code

References

[1] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, volume 49, pages 269–284. ACM, 2014.

[2] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 367–379. IEEE, 2016.

[3] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014.

[4] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 92–104. ACM, 2015.

[5] Vinayak Gokhale, Jonghoon Jin, Aysegul Dundar, Berin Martini, and Eugenio Culurciello. A 240 g-ops/s mobile coprocessor for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 682–687, 2014.

[6] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Proteus: Exploiting numerical precision variability in deep neural networks. In *Proceedings of the 2016 International Conference on Supercomputing*, page 23. ACM, 2016.

[7] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, and Andreas Moshovos. Stripes: Bit-serial deep neural network computing. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pages 1–12. IEEE, 2016.

[8] Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pages 553–564. IEEE, 2017.

[9] Bert Moons, Roel Uytterhoeven, Wim Dehaene, and Marian Verhelst. Envision: A 0.26-to-10 tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 246–257, 2017.

[10] Lili Song, Ying Wang, Yinhe Han, Xin Zhao, Bosheng Liu, and Xiaowei Li. C-brain: A deep learning accelerator that tames the diversity of cnns through adaptive data-level parallelization. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2016.

[11] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel Emer. Efficient processing of deep neural networks: A tutorial and survey. *arXiv preprint arXiv:1703.09039*, 2017.

[12] Hokchhay Tann, Soheil Hashemi, R Iris Bahar, and Sherief Reda. Hardware-software codesign of accurate, multiplier-free deep neural networks. In *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*, pages 1–6. IEEE, 2017.

[13] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74. ACM, 2017.