论文选读0628

2019 年 6 月 28 日

1 Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding

1.1 Abstract

1.2 Network Pruning

不够详细需要解释

2 Learning both Weights and Connections for Efficient Neural Networks

3 EIE: Efficient Inference Engine on Compressed Deep Neural Network

使用compressed network.

从DRAM到完全SRAM(120X energy saving); 利用矩阵稀疏性(10X); 权值共享(8X); 跳过ReLU零激活(3X).

文章贡献:
1. 呈现了第一个为稀疏的、权值共享的神经网络准备的加速器.直接在压缩神经网络上运算使得大的神经网络模型可以放到片上的SRAM上，与向外部的DRAM 取数相比节省120X的能量.
2. EIE是第一个利用了激活函数的动态稀疏性以减少计算的加速器. EIE避免了百分之70的零激活的引用和算术，因此节省了百分之65.15的能量.
3.描述了一种针对分布式存储和分布式计算的方法: 在多PE之间并行化一个稀疏层，达到load balance和good scalability.
4.在大量的深度学习模型上评估EIE，包含物体检测的CNN，自然语言处理和图片抓取的LSTM.比较了EIE和CPU，GPU，FPGA和其它ASIC加速器.

3.1 DNN Compression And Parallelization

3.1.1 Computation

$$b_i = ReLU(\sum_{j \in X_i \cap Y} S[I_{ij}]a_j)$$

X_i是表示列j中$W_{ij} \neq 0$的集合, Y是表示索引j满足$a_j \neq 0$的集合. S 是共享权值表, I_{ij}是对共享权值的索引.
这造成了动态的非规律计算, 进行索引造成了额外的load操作，但它几乎是可以cache hit 的(为什么?).

3.1.2 Representation

稀疏权值矩阵W被以可变的压缩稀疏列(**CSC** for compressed sparse column) 的形式存储. 对矩阵W的每一列: 存储一个向量v, 它包含了非零的权值；存储一个相同长度的向量z, 表示对应v中的项之前的0的个数. v和z 的项都用一个4bit值表示. 如果一个非0的项之前有超过15个0，那么在向量v里增加一个0.举例:编码列$[0,0,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,\mathbf{0},0,0,3]$为$v = [1,2,\mathbf{0},3]$, $z = [2,0,\mathbf{15},2]$. 每一列的v和z都被存储到一个大的数组对中，有一个指针向量p只想每一列的向量的开始处.p的最后一项比最后一个向量元素大1，以使列j的非零元素(包括padded zeros) 可以通过$p_{j+1} - p_j$计算出来.
以按列CSC的格式存储系数矩阵使得能更容易地利用激活变量的稀疏性.

3.1.3 Parallelizing Compressed DNN

矩阵分布式存储、并行化矩阵向量计算的方式是将矩阵W的行分划到多个处理单元上(**PEs**).若有N个PEs，那么PE_k存储的是所有满足$i \bmod N = k$的行W_i, 输出激活量b_i和输入激活量a_i. PE_k中列W_i 的部分被以CSC格式存储, 其中零的计数仅仅计在该列在PE_k的子集的零.每个PE都有自己的v, x和p序列.

稀疏矩阵与稀疏向量的乘法是这样进行的: 扫描向量a找到下一个非零值a_j, 将a_j 和系数j广播到所有的PE中，每个PE用其列W_j中非零的元素乘以a_j, 累加到向量b中.在CSC表示法中，非零权值是连续存储的，因此每个PE 只要遍历它的v序列从p_j到$p_{j+1} - 1$的位置以加载权值.

划分式的CSC表示法利用了激活向量a的动态稀疏性和权值矩阵W的静态稀疏性. 这种组织方式使得除了广播外的计算对PE来说是local的. 这个过程可能会有*load imbalance*的问题因为某一列中每个PE可能有不同数量的非零元素. 这可以通过*queueing*解决.

3.2 Hardware Implementation

EIE的架构是: 有一个中央控制单元(**CCU** short for **C**entral **C**ontrol **U**nit), 它控制了PE序列. CCU从一个分布式的前导零探测网络接受非零激活输入，并将它们广播到所有的PEs中.

EIE中几乎所有的计算都是local to PEs的, 除了收集并广播非零的激活输入. 但是收集和广播是非关键(non-critical)的, 因为大部分的PEs对每个激活输入都要耗上很多周期.

3.2.1 Activation Queue and Load Balancing

输入激活向量的非零元素a_j和对应的索引j被CCU广播到每个PE中的激活队列. 如果任何一个PE中队列已满，那么广播被disabled. 在任意时间点，每个PE处理队列头的激活元素.

激活队列使得每个PE建立一个工作积压区以平衡可能出现的load imbalance.

3.2.2 Pointer Read Unit

激活队列头部的项的索引j被用来查找针对列j的序列v和x的开始和结束的指针p_j 和p_{j+1}. 为使这两个指针可以在一个周期内从一个单口SRAM读出, EIE将指针存在两个SRAM banks里, 按照LSB在bank间选择. p_j和p_{j+1}将始终在不同的bank中. EIE的指针是16bit长的.

3.2.3 Sparse Matrix Read Unit

稀疏矩阵读取单元使用指针P_j和p_{j+1}从稀疏矩阵SRAM中读取此PE切片列I_j(对W_j的索引)的非零元素. SRAM中的每一项都是8bit的, 包含v 的一个4bit(共享表S有16个元素)元素和x 的一个4bit元素.

效率起见, PE切片的编码稀疏矩阵I被存储在一个64bit宽的SRAM中. 因此每个SRAM读可以取得8个表项. 当前指针p的高13bit选择了一个SRAM行，低3bit从该行的8个表项中选择一项. 每个周期，一个(v,x)项被提供给算术单元(Arithmetic Unit).

3.2.4 Arithmetic Unit

算术单元从稀疏矩阵读取单元(Sparse Matrix Read Unit)中接收一个(v,x)表项, 执行乘加操作: $b_x = b_x + v \times a_j$. x用来索引一个累加序列, v事先通过查表被扩展成一个16bit定点数. 如果在相邻的两个周期同一个累加器都被选中，那么可以走旁路(bypassing).

3.2.5 Activation Read/Write

激活读写单元包含两个激活寄存器以在单轮FC层计算中分别处理源与目的激活值. 源和目的寄存器在下一层交换他们的角色. 因此在多层前向的计算中没有额外的数据传输.

每个激活寄存器有64个16bit的激活值. 这对在64个PE中处理4K的激活向量来说是足够的. 更长激活向量可以被2KB的激活SRAM处理. 当激活向量的长度大于4K时, $M \times V$会被分批(batch)完成. 每一批的长度是4K或更少. 所有的局部生产都是在寄存器里完成的. SRAM只在每batch的开始

时被读，每个batch 结束时被写.(这种情况下后一层交换角色有无意义？)

3.2.6 Distributed Leading Non-Zero Detection

输入激活量是分层传给每个PE的, 为了利用输入向量的稀疏性, EIE使用了前导非零探测逻辑以选择第一个非零结果. 每个4 PE的组在它们的输入激活量的基础上进行一个局部的前导非零探测逻辑. 结果被送到前导非零探测节点(LNZD节点) 上. 每个LNZD节点在它的四个children中寻找下一个非零的激活量,并将结果传给四叉树. 在LNZD根节点，选中的非零激活值被通过一个Htree 内的线路广播回所有的PEs.

3.2.7 Central Control Unit

中央控制单元是根LNZD节点. 它和master通信(比如CPU),并且通过设置控制寄存器监管每个PE的状态. 有两种模式: I/O 和计算. 在I/O模式中，所有的PE都是空闲的, PEs内的激活量和权值都可以被CCU通过与其(CCU)相连的DMA获取. 在计算模式中, CCU重复地从LNZD四叉树中收集非零值并把它们广播道所有的PEs, 这个阶段持续进行直到超过了输入长度. 通过设置输入长度和指针序列的起始位置, EIE可以处理不同的层.

4 Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing

5 Cambricon-X: An Accelerator for Sparse Neural Networks

6 SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks

文章介绍了**SCNN**(short for **S**parse **CNN**)加速器架构. 它是一个新的CNN推断架构，利用了权值和激活量的稀疏性以提升DNN 的性能, 减少功耗. SCNN加速器被用来优化卷积层的计算，因为设计良好的处理计算机图像的DNN主要被这些计算密集的层主导.

SCNN是第一个有效地同时处理无效激活量和权值的稀疏CNN加速器. 此外，SCNN应用了1.算术数据流(algorithmic dataflow)以消减所有含零的乘法; 2.权值和激活量的压缩表示(几乎在整个计算过程中都是如此).

SCNN的设计核心是处理单元(PE), 它有一个乘法阵列，接收权值数组和激活量数组. 与之前的卷积层不同, SCNN数据流只将全部可以做笛卡尔乘法的权值和激活量传给乘法阵列. 为了减少数据的读取, 激活向量在与一系列的权值向量相乘的过程中是以输入站(input stationary)的形式被重用的. 只有非零的权值和激活量会被从输入存储阵列中取出并送到乘法阵列.

因为由乘法阵列生成的乘积不能被直接相加, SCNN追踪与每个乘积相关的**输出系数(?)**，并把系数和成绩传递给一个用来做加法的散列加法阵列.

多个PE可以并行运行, 每个都在输入激活量的一组不相关的3D **tile(?)**上运行. CNN数据的压缩和tiling使得两个能量节约的优化成为可能: 首先, 在流水线中以压缩的形式保持权值和激活量减少了耗能的data staging和传输消耗; 其次, 整个大CNN的激活量在层与层之间可以留在on-die 上的缓冲, 消除了大型网络昂贵的跨层DRAM引用. 总体来说, 这个设计提供了有效的压缩存储和输入操作数传递, 利用了乘法阵列中输入操作数的高重用性，并且在零操作数的乘法上不费任何时间.

6.1 Motivation

CNN本质上是需要被训练的级联模式识别过滤器组.一个CNN包含一系列的层，包含卷积层，非线性标量运算器层和采取诸如池化的方式对中间数据降低取样的层. 卷积层代表了CNN计算的核心并以一组过滤器为特征, 它们通常是1X1或3X3的, 但有时也是5X5或更大的. 这些过滤器的值是通过一组网络的训练集学到的权值. 一些深度神经网络也包含全相连层, 主要在DNN的结束部分.

在推断阶段, 一个新图片被传给网络, 通过相继计算网络中的各层将它分到一个训练类别中. 论文主要关注卷积层的加速，因为它们占据着计算的大部分.

6.1.1 Sparsity in CNNs

CNN层的稀疏性定义为此层权值和输入激活量矩阵中零的比例. 达到权值稀疏性的基本方法是在训练过程中对网络进行剪枝. 首先, 所有绝对值接近零的权值被设成零. 这个过程的效果是将权值从过滤器中除去, 有时甚至迫使一个输出激活量变成恒零. 其次, 剩下的网络被重新训练以重新提升在朴素剪枝中丢掉的准确度. 得到的结果是一个与原先网络的准确度极为接近的更小的网络. 这个过程可以被重复迭代以在保持准确度的同时减小网络的尺寸.

激活量的稀疏性是在推断过程中动态产生的, 并且高度依赖于正在被处理的数据. 特别地, ReLU 函数迫使负的量变成零. 在完成一层卷积层的计算后, ReLU函数在数据被传给下一层之前对输出激活量矩阵中的每个元素逐点(**point-wise**)运算.

6.1.2 Exploiting sparsity

由于含零的乘法的结果还是零, 它不需要处理. 因此典型的层能以4倍减少工作量并可能达到将10 倍. 此外这些零的结果对部分和不做任何贡献, 所以加法也是不必要的. 另外, 有许多零的数据可以被以压缩的形式表示. 这些性质合起来可以为优化提供许多机会:
1.压缩数据.将系数权值和激活量压缩提供给架构减少必须通过memory层移动的数据的数目的机会.它也减少了数据的面积，因此可以使给定大小的存储结构存下更大的矩阵.
2.减少计算量. 对含零的乘法, 操作可以是数据门控的(data gated), 或者操作数永远不被送到乘法器. 这个优化节省了电量消耗或分别减少了时间和电量的消耗.

SCNN的目标是利用激活量和剪枝权值的稀疏性以减少尽可能多的计算周期和数据移动、存储操作. SCNN采用了稀疏权值和激活量的密集编码以使得只有非零数值才会从DRAM和片上buffer中被取出.不幸的是, 组织一个数据流在最大化数据重用和乘法器利用率的过程中将这些稀疏数据集传给乘法器阵列不是平凡的.在只将非零权值和激活量传给乘法器的同时，SCNN采用了一个创新的笛卡尔乘积数据流阵列. 这个数据流对非零权值向量和激活向量元素执行**all to all(?)**的乘法, 以避免任何基于零值操作数的计算并在稳态达到完全的乘法器利用率.

6.2 SCNN Architecture

6.2.1 Tiled Architecture

PE与他们的最近邻居互联以在处理过程中交换*halo value*. PE array是被一个*layer sequence*驱动的，以组织权值和激活量的移动; 它也和DRAM controller相连，以向PE广播权值和stream activations. SCNN可以用仲裁总线作为全局网络以帮助权值的广播、从DRAM来的点对点的输入激活量的传输和向DRAM写回输出激活量.

6.2.2 Processing Element Architecture

一个PE包含: 权值缓存(weight buffer), 输入输出激活RAM(IARAM and OARAM short for input/output activation RAMs), 乘法阵列(multiplier array), 散列交叉开关(scatter crossbar), 累加缓存bank(bank of accumulator buffer)和后处理单元(PPU short for post processing unit).为了处理第一层的CNN, layer sequencer将输入图片的一部分流(stream)进每个PE的IARAM中，并将压缩稀疏权值广播道每个PE的权值缓存中, 完成该层的计算时, 压缩稀疏的输出激活量被分布到PE的OARAM中. 如果可能, 激活量一直在IARAM/OARAM中, 从未被换出到DRAM 中. 如果一层的输出权值可以作为下一层的输入权值, 那么IARAMs和OARAMs在两层间的计算阵列中是在逻辑上进行交换的. CNN的每层都有一组参数配置layer sequencer中的controller, weight FIFO, IARAMs/OARAMs和PPU以进行需要的计算.

Input weights and activations.每个PE的状态机按照**PT-IS-CP**稀疏数据流定义的顺序对weight和输入激活量进行操作以缠上一个输出通道组的部分和.首先一个压缩权值向量F和一个压缩输入激活量I被从它们各自的缓存中取出. 这些向量被分布到$F \times I$的乘法阵列中以进行向量间的笛卡尔乘积. 同时稀疏压缩权值和激活量的索引被处理以计算在密集输出空间的输出坐标.

Accumulation.$F \times I$乘积被传输给一列A长的累加器bank, 它是被输出坐标索引的. 为了减少被hash到同一个累加bank的乘积之间的contention, A被设定成大于$F \times I$, 实验结果表明$A = 2 \times F \times I$足够减少累加器的bank contention。每个累加器缓冲包含加法器和一小组与正在被处理的输出通道组关联的输出通道表项. 累加器缓冲是双缓冲**?** 的, 因此一组bank被输入部分和更新的同时另一组bank可以被PPU消耗.

Post-processing.当输出通道组完成后, PPU执行以下的任务:(1)和邻近的PE交换处在PE 输出激活量边界的halo region的部分和.(2)应用非线性激活(例如ReLU), pooling和dropout functions.(3)将输出激活量压缩成compressed-sparse形式，写进OARAM中. 除了neighbour halo exchange，这些操作被限制在该PE自己产生的数据值上.

Compression.为了压缩权值和激活量, 采用了以往工作(EIE)提及的稀疏矩阵表示方法.

6.2.3 Fully-connected Layers

SCNN对全连接层的处理并不好.

6.2.4 Temporal Tiling for Large Models

有些层的激活量需要被存到DRAM中. SCNN可以在时序上tile the activation space使得在同一时刻, PE们在激活量的子集上工作.

DRAM的访存消耗可以通过和另一个tile的并行加上流水化的形式隐藏掉.

7 ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA