北京大学
PEKING UNIVERSITY

智能硬件体系结构

第十二讲：人工智能加速器架构（II）

主讲：陶耀宇、李萌

2024年秋季

注意事项

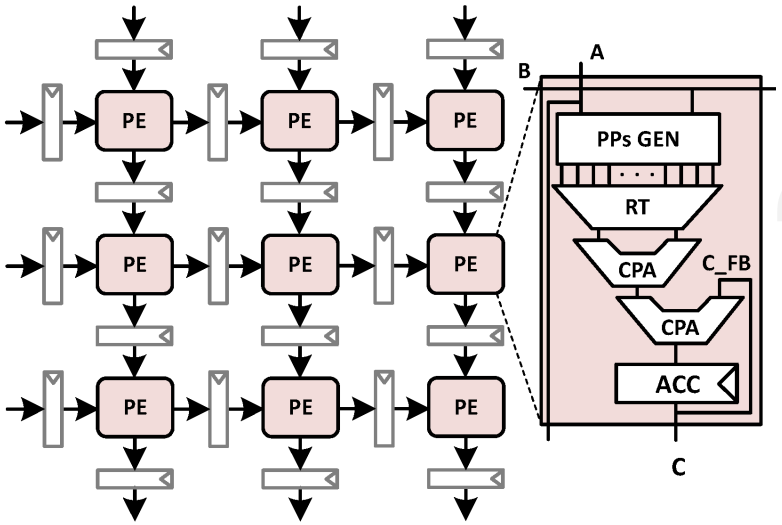
北京大学 PEKING UNIVERSITY

- **课程作业情况**

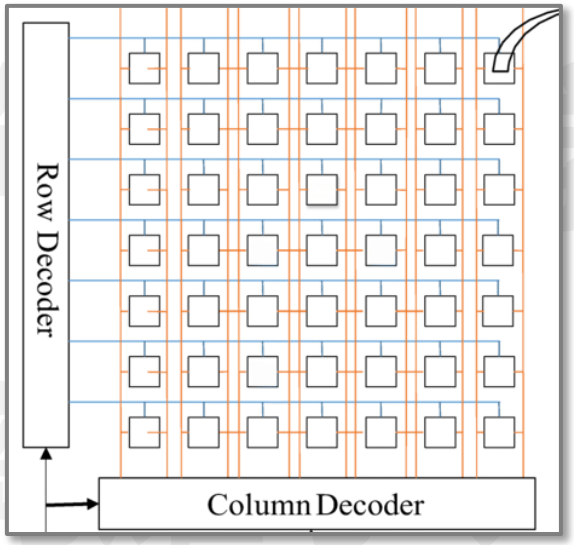
  - 未来**1次作业（12.15-12.30）**
    - AI芯片架构、软硬加协同优化
  - **Paper Review（12月25号/27号1-3点，每个同学10分钟）**
    - 时间有疑义请尽快联系老师或助教提出更换
    - 请自行阅读近5年内的体系结构相关论文1-2篇
    - 做8页左右的PPT，**每个同学10分钟演讲 + 1~2分钟答疑**
  - 第二次实验已经上传到课程网站，截止日期为**12月31号**

AI加速器架构基础-Recap

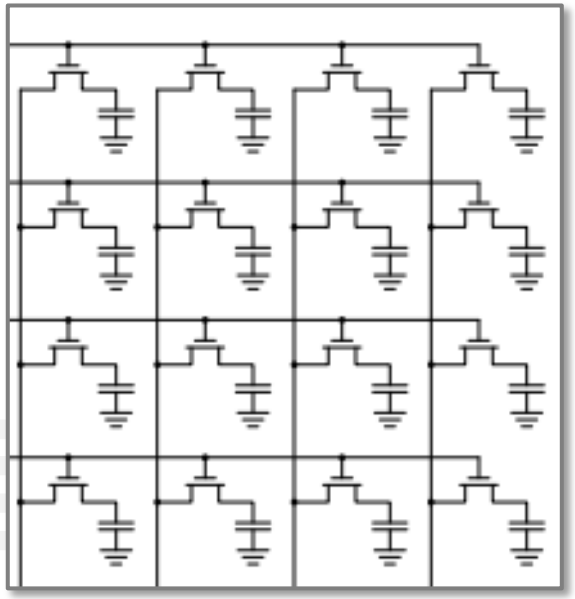
- 上次课，我们介绍了AI加速器的基础架构，并着重介绍了AI加速器中的计算单元、片上访存以及片间/片外通信
- 本节课我们将介绍有代表性的**AI加速器架构**



On-chip SRAM



Off-chip DRAM



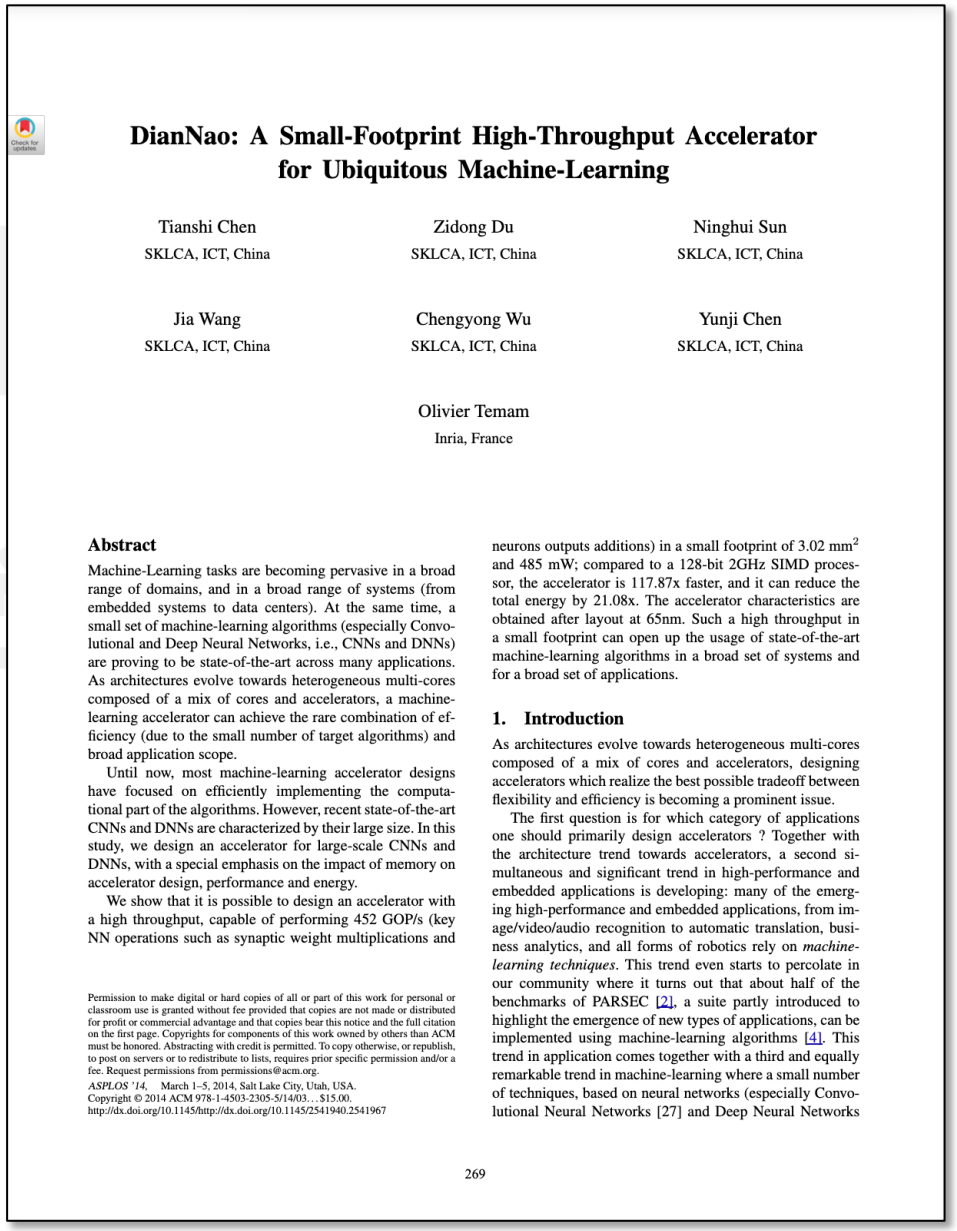
计算　　访存　　通信

AI加速器架构发展——DianNao

北京大学 PEKING UNIVERSITY

- **ASPLOS 2014最佳论文**
- **DianNao对于神经网络加速器设计进行了系统讨论**
  - **如何设计计算阵列?**
  - **如何设计片上存储?**
  - **如何提升数据复用，降低访存开销?**
  - **如何灵活支持不同操作?**
- **DianNao面向MLP和CNN，相关参数如下**

Performance	452 GOPS
Area	3.02 mm2
Power	485 mW
Technology Node	65 nm
Baseline	117.87X speedup over SIMD processor



DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning

Tianshi Chen, SKLCA, ICT, China
Zidong Du, SKLCA, ICT, China
Ninghui Sun, SKLCA, ICT, China
Jia Wang, SKLCA, ICT, China
Chengyong Wu, SKLCA, ICT, China
Yunji Chen, SKLCA, ICT, China
Olivier Temam, Inria, France

Abstract

Machine-Learning tasks are becoming pervasive in a broad range of domains, and in a broad range of systems (from embedded systems to data centers). At the same time, a small set of machine-learning algorithms (especially Convolutional and Deep Neural Networks, i.e., CNNs and DNNs) are proving to be state-of-the-art across many applications. As architectures evolve towards heterogeneous multi-cores composed of a mix of cores and accelerators, a machine-learning accelerator can achieve the rare combination of efficiency (due to the small number of target algorithms) and broad application scope.

Until now, most machine-learning accelerator designs have focused on efficiently implementing the computational part of the algorithms. However, recent state-of-the-art CNNs and DNNs are characterized by their large size. In this study, we design an accelerator for large-scale CNNs and DNNs, with a special emphasis on the impact of memory on accelerator design, performance and energy.

We show that it is possible to design an accelerator with a high throughput, capable of performing 452 GOP/s (key NN operations such as synaptic weight multiplications and neurons outputs additions) in a small footprint of 3.02 mm^2 and 485 mW; compared to a 128-bit 2GHz SIMD processor, the accelerator is 117.87x faster, and it can reduce the total energy by 21.08x. The accelerator characteristics are obtained after layout at 65nm. Such a high throughput in a small footprint can open up the usage of state-of-the-art machine-learning algorithms in a broad set of systems and for a broad set of applications.

ASPLOS '14, March 1–5, 2014, Salt Lake City, Utah, USA.

http://dx.doi.org/10.1145/http://dx.doi.org/10.1145/2541940.2541967

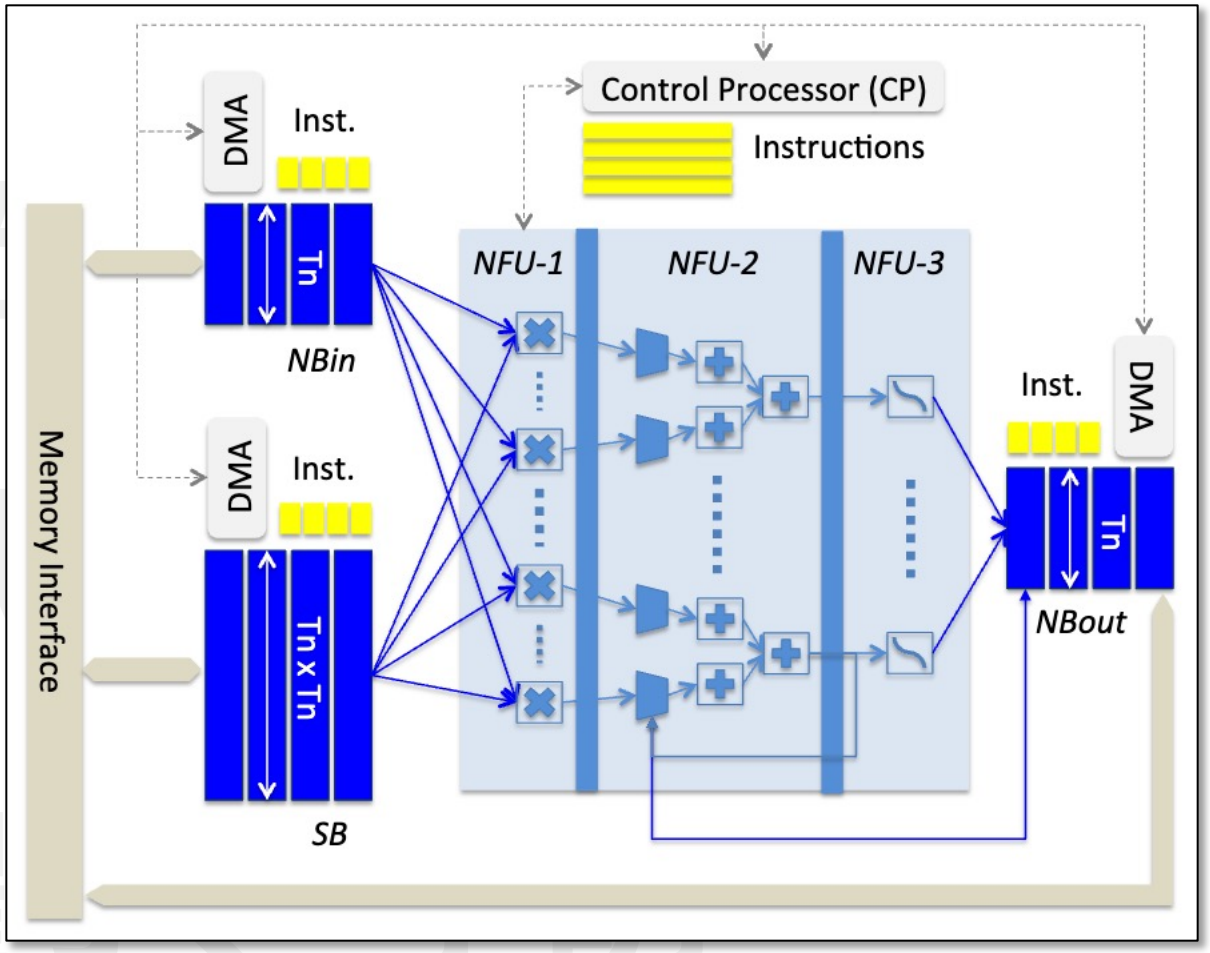
1. Introduction

As architectures evolve towards heterogeneous multi-cores composed of a mix of cores and accelerators, designing accelerators which realize the best possible tradeoff between flexibility and efficiency is becoming a prominent issue.

The first question is for which category of applications one should primarily design accelerators ? Together with the architecture trend towards accelerators, a second simultaneous and significant trend in high-performance and embedded applications is developing: many of the emerging high-performance and embedded applications, from image/video/audio recognition to automatic translation, business analytics, and all forms of robotics rely on *machine-learning techniques*. This trend even starts to percolate in our community where it turns out that about half of the benchmarks of PARSEC [2], a suite partly introduced to highlight the emergence of new types of applications, can be implemented using machine-learning algorithms [4]. This trend in application comes together with a third and equally remarkable trend in machine-learning where a small number of techniques, based on neural networks (especially Convolutional Neural Networks [27] and Deep Neural Networks

AI加速器架构发展——DianNao

- 如何进行神经网络加速器设计?
  - 直接实例化完整神经网络 vs 通用神经网络计算架构实现多算子支持
- 如何进行计算阵列设计?
- 如何设计访存?
  - High associative cache vs 便签存储器



AI加速器架构发展——DianNao

```
for (int nnn = 0; nnn ¡ Nn; nnn += Tnn) { // tiling for output neurons;
  for (int iii = 0; iii ¡ Ni; iii += Tii) { // tiling for input neurons;
    for (int nn = nnn; nn ¡ nnn + Tnn; nn += Tn) {
      for (int n = nn; n ¡ nn + Tn; n++)
        sum[n] = 0;
      for (int ii = iii; ii ¡ iii + Tii; ii += Ti)
        // — Original code —
        for (int n = nn; n < nn + Tn; n++)
          for (int i = ii; i < ii + Ti; i++)
            sum[n] += synapse[n][i] * neuron[i];
      for (int n = nn; n < nn + Tn; n++)
        neuron[n] = sigmoid(sum[n]);
} } }
```

全连接层

```
for (int yy = 0; yy ¡ Nyin; yy += Ty) {
  for (int xx = 0; xx ¡ Nxin; xx += Tx) {
    for (int nnn = 0; nnn ¡ Nn; nnn += Tnn) {
      // — Original code — (excluding nn, ii loops)
      int yout = 0;
      for (int y = yy; y < yy + Ty; y += sy) { // tiling for y;
        int xout = 0;
        for (int x = xx; x < xx + Tx; x += sx) { // tiling for x;
          for (int nn = nnn; nn < nnn + Tnn; nn += Tn) {
            for (int n = nn; n < nn + Tn; n++)
              sum[n] = 0;
            // sliding window;
            for (int ky = 0; ky < Ky; ky++)
              for (int kx = 0; kx < Kx; kx++)
                for (int ii = 0; ii < Ni; ii += Ti)
                  for (int n = nn; n < nn + Tn; n++)
                    for (int i = ii; i < ii + Ti; i++)
                      // version with shared kernels
                      sum[n] += synapse[ky][kx][n][i]
                                * neuron[ky + y][kx + x][i];
                      // version with private kernels
                      sum[n] += synapse[yout][xout][ky][kx][n][i]}
                                * neuron[ky + y][kx + x][i];
            for (int n = nn; n < nn + Tn; n++)
              neuron[yout][xout][n] = non_linear_transform(sum[n]);
          } xout++; } yout++;
} } } }
```

卷积层

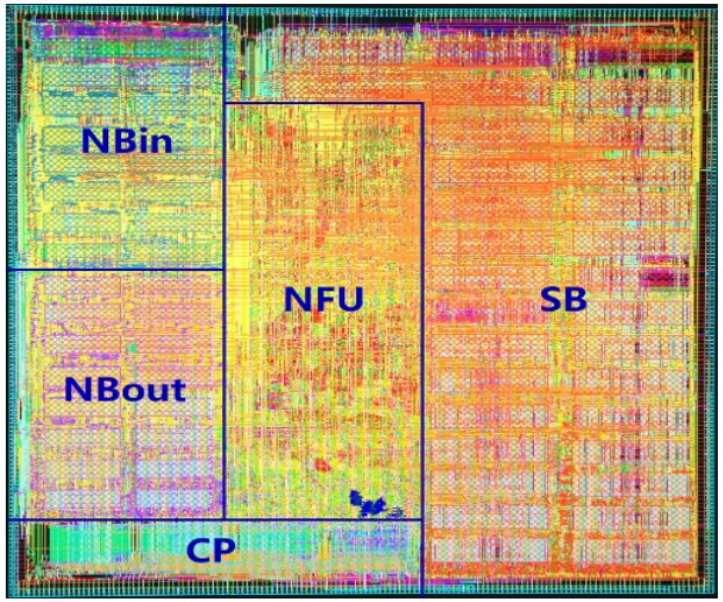
AI加速器架构发展——DianNao

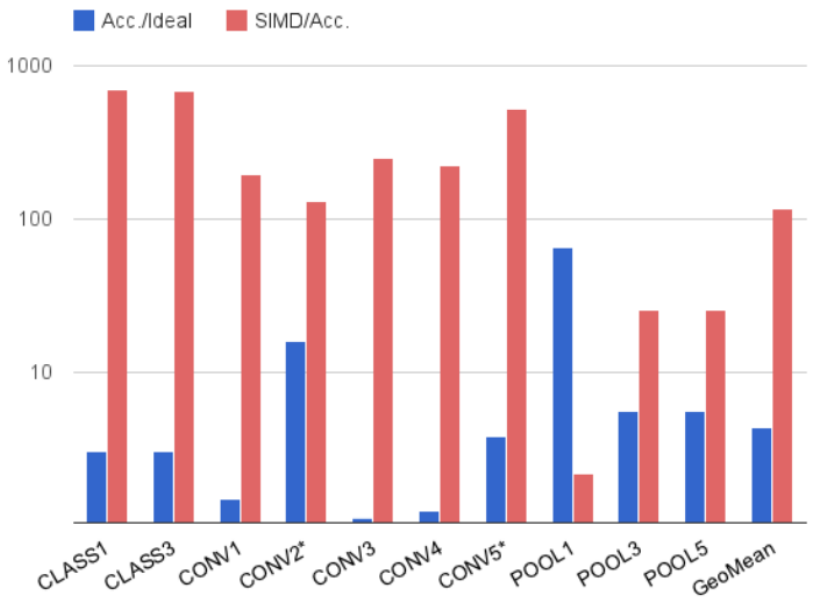
- 如何处理池化层?
  - 池化层和卷积层具有不同的访存需求
  - 通过局部转置的方式，在DMA读入输入激活值的时候，采用interleaving的方式，将输入激活值存储在便签存储器中
- 这种设计有没有什么问题?

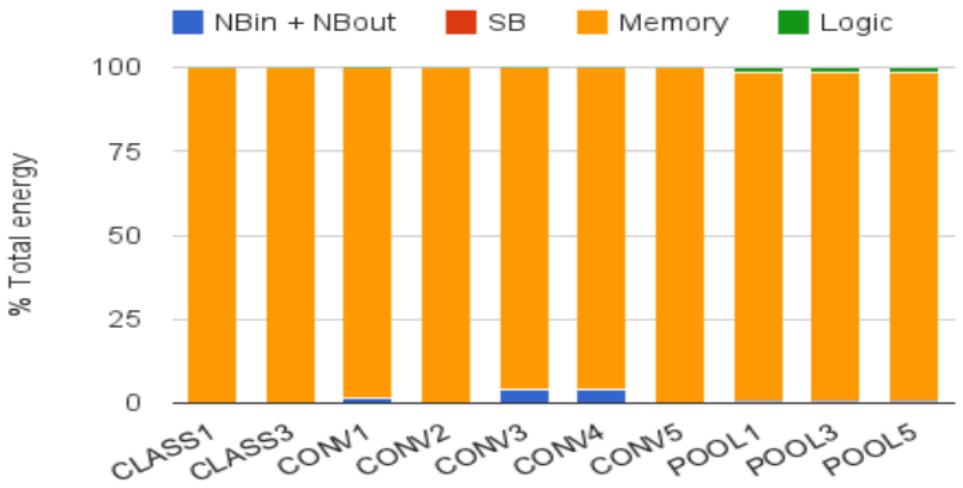
```
for (int yy = 0; yy ¡ Nyin; yy += Ty)  {
  for (int xx = 0; xx ¡ Nxin; xx += Tx)  {
    for (int iii = 0; iii ¡ Ni; iii += Tii)
      // — Original code — (excluding ii loop)
      int yout = 0;
      for (int y = yy; y < yy + Ty; y += sy) {
        int xout = 0;
        for (int x = xx; x < xx + Tx; x += sx) {
          for (int ii = iii; ii < iii + Tii; ii += Ti)
            for (int i = ii; i < ii + Ti; i++)
              value[i] = 0;
            for (int ky = 0; ky < Ky; ky++)
              for (int kx = 0; kx < Kx; kx++)
                for (int i = ii; i < ii + Ti; i++)
                  // version with average pooling;
                  value[i] += neuron[ky + y][kx + x][i];
                  // version with max pooling;
                  value[i] = max(value[i], neuron[ky + y][kx + x][i]);
        } } } }
        // for average pooling;
        neuron[xout][yout][i] = value[i] / (Kx * Ky);
        xout++; } yout++;
} } }
```

AI加速器架构发展——DianNao

- 相较于CPU的SIMD计算，实现了显著的计算速度提升
- 但是，相较于理论加速比，仍然存在一定的差距，即硬件仍然面临欠利用问题
- 主要功耗来源于DRAM的访问

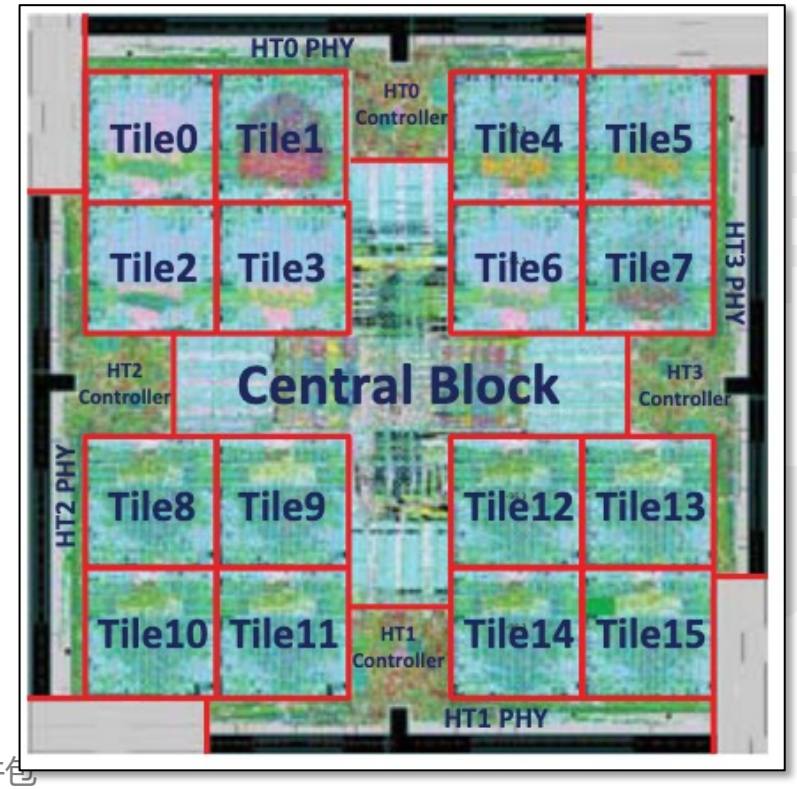






AI加速器架构发展——DaDianNao

- **MICRO 2014最佳论文**
- **Multi-chip神经网络加速器：机器学习超级计算机**



DaDianNao: A Machine-Learning Supercomputer

Yunji Chen[1], Tao Luo[1,3], Shaoli Liu[1], Shijin Zhang[1], Liqiang He[2,4], Jia Wang[1], Ling Li[1], Tianshi Chen[1], Zhiwei Xu[1], Ninghui Sun[1], Olivier Temam[2]

[1] SKL of Computer Architecture, ICT, CAS, China
[2] Inria, Scalay, France
[3] University of CAS, China
[4] Inner Mongolia University, China

***Abstract*—Many companies are deploying services, either for consumers or industry, which are largely based on machine-learning algorithms for sophisticated processing of large amounts of data. The state-of-the-art and most popular such machine-learning algorithms are Convolutional and Deep Neural Networks (CNNs and DNNs), which are known to be both computationally and memory intensive. A number of neural network accelerators have been recently proposed which can offer high computational capacity/area ratio, but which remain hampered by memory accesses.**

However, unlike the memory wall faced by processors on general-purpose workloads, the CNNs and DNNs memory footprint, while large, is not beyond the capability of the on-chip storage of a multi-chip system. This property, combined with the CNN/DNN algorithmic characteristics, can lead to high internal bandwidth and low external communications, which can in turn enable high-degree parallelism at a reasonable area cost. In this article, we introduce a custom multi-chip machine-learning architecture along those lines. We show that, on a subset of the largest known neural network layers, it is possible to achieve a speedup of 450.65x over a GPU, and reduce the energy by 150.31x on average for a 64-chip system. We implement the node down to the place and route at 28nm, containing a combination of custom storage and computational units, with industry-grade interconnects.

I. Introduction

Machine-Learning algorithms have become ubiquitous in a very broad range of applications and cloud services; examples include speech recognition, e.g., Siri or Google Now, click-through prediction for placing ads [27], face identification in Apple iPhoto or Google Picasa, robotics [20], pharmaceutical research [9] and so on. It is probably not exaggerated to say that machine-learning applications are in the process of displacing scientific computing as the major driver for high-performance computing. Early symptoms of this transformation are Intel calling for a refocus on Recognition, Mining and Synthesis applications in 2005 [14] (which later led to the PARSEC benchmark suite [3]), with Recognition and Mining largely corresponding to machine-learning tasks, or IBM developing the Watson supercomputer, illustrated with the Jeopardy game in 2011 [19].

Remarkably enough, at the same time this profound shift in applications is occurring, two simultaneous, albeit apparently unrelated, transformations are occurring in the machine-learning and in the hardware domains. Our community is well aware of the trend towards heterogeneous computing where architecture specialization is seen as a promising path to achieve high performance at low energy [21], provided we can find ways to reconcile architecture specialization and flexibility. At the same time, the machine-learning domain has profoundly evolved since 2006, where a category of algorithms, called Deep Learning (Convolutional and Deep Neural Networks), has emerged as state-of-the-art across a broad range of applications [33], [28], [32], [34]. In other words, at the time where architects need to find a good tradeoff between flexibility and efficiency, it turns out that just one category of algorithms can be used to implement a broad range of applications. In other words, there is a fairly unique opportunity to design highly specialized, and thus highly efficient, hardware which will benefit many of these emerging high-performance applications.

A few research groups have started to take advantage of this special context to design accelerators meant to be integrated into heterogeneous multi-cores. Temam [47] proposed a neural network accelerator for multi-layer perceptrons, though it is not a deep learning neural network, Esmaeilzadeh et al. [16] propose to use a hardware neural network called NPU for approximating any program function, though not specifically for machine-learning applications, Chen et al. [5] proposed an accelerator for Deep Learning (CNNs and DNNs). However, all these accelerators have significant neural network *size* limitations: either small neural networks of a few tens of neurons can be executed, or the neurons and synapses (i.e., weights of connections between neurons) intermediate values have to be stored in main memory. These two limitations are severe, respectively from a machine-learning or a hardware perspective.

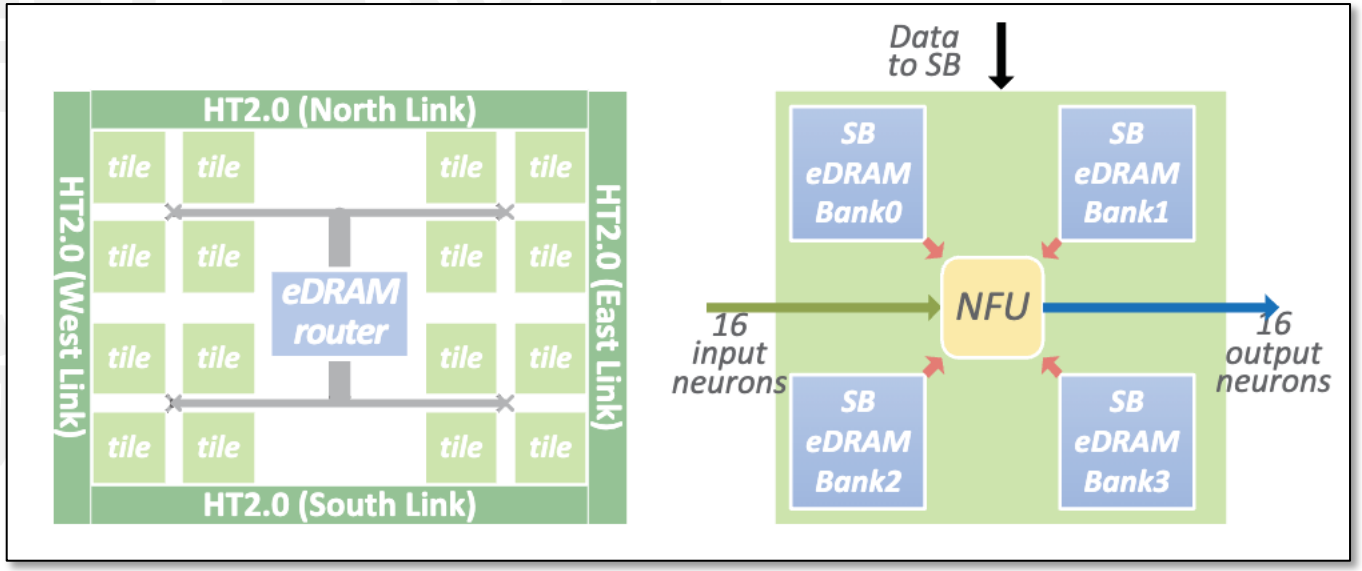
From a machine-learning perspective, there is a significant trend towards increasingly large neural networks. The recent work of Krizhevsky et al. [32] achieved state-of-the-art accuracy on the ImageNet database [13] with "only" 60

AI加速器架构发展——DaDianNao

- DaDianNao的设计特点：
  - 分块（tile）设计，模型权重存储靠近计算单元，采用4-bank eDRAM进行存储
  - 采用胖树的路由设计，进行输入和输出激活值的广播
  - 芯片中间设置2个eDRAM bank，进行输入和输出激活值的存储
  - 主要针对MLP和CNN设计，同时支持训练和推理

Parameters	Settings	Parameters	Settings
Frequency	606MHz	tile eDRAM latency	~3 cycles
# of tiles	16	central eDRAM size	4MB
# of 16-bit multipliers/tile	256+32	central eDRAM latency	~10 cycles
# of 16-bit adders/tile	256+32	Link bandwidth	6.4x4GB/s
tile eDRAM size/tile	2MB	Link latency	80ns

Table III: *Architecture characteristics.*



AI加速器架构发展——Eyeriss

北京大学 PEKING UNIVERSITY

- 2016年，MIT团队相继在HPCA、ISSCC、JSSC发表论文Eyeriss，对于学术界产生了重要影响
- Eyeriss在对于不同数据流进行系统比较的基础上，提出了新的row stationary数据流，并流片验证

Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks

Yu-Hsin Chen*, Joel Emer*† and Vivienne Sze*

*EECS, MIT
Cambridge, MA 02139

†NVIDIA Research, NVIDIA
Westford, MA 01886

*{yhchen, jsemer, sze}@mit.edu

***Abstract*—Deep convolutional neural networks (CNNs) are widely used in modern AI systems for their superior accuracy but at the cost of high computational complexity. The complexity comes from the need to simultaneously process hundreds of filters and channels in the high-dimensional convolutions, which involve a significant amount of data movement. Although highly-parallel compute paradigms, such as SIMD/SIMT, effectively address the computation requirement to achieve high throughput, energy consumption still remains high as data movement can be more expensive than computation. Accordingly, finding a dataflow that supports parallel processing with minimal data movement cost is crucial to achieving energy-efficient CNN processing without compromising accuracy.**

In this paper, we present a novel dataflow, called *row-stationary* (RS), that minimizes data movement energy consumption on a spatial architecture. This is realized by exploiting local data reuse of filter weights and feature map pixels, i.e., activations, in the high-dimensional convolutions, and minimizing data movement of partial sum accumulations. Unlike dataflows used in existing designs, which only reduce certain types of data movement, the proposed RS dataflow can adapt to different CNN shape configurations and reduces all types of data movement through maximally utilizing the processing engine (PE) local storage, direct inter-PE communication and spatial parallelism. To evaluate the energy efficiency of the different dataflows, we propose an analysis framework that compares energy cost under the same hardware area and processing parallelism constraints. Experiments using the CNN configurations of AlexNet show that the proposed RS dataflow is more energy efficient than existing dataflows in both convolutional (1.4× to 2.5×) and fully-connected layers (at least 1.3× for batch size larger than 16). The RS dataflow has also been demonstrated on a fabricated chip, which verifies our energy analysis.

I. Introduction

The recent popularity of deep learning [1], specifically deep convolutional neural networks (CNNs), can be attributed to its ability to achieve unprecedented accuracy for tasks ranging from object recognition [2–5] and detection [6, 7] to scene understanding [8]. These state-of-the-art CNNs [2–5] are orders of magnitude larger than those used in the 1990s [9], requiring up to hundreds of megabytes for filter weight storage and 30k-600k operations per input pixel.

The large size of such networks poses both throughput and energy efficiency challenges to the underlying processing hardware. Convolutions account for over 90% of the CNN operations and dominates runtime [10]. Although these operations can leverage highly-parallel compute paradigms, such as SIMD/SIMT, throughput may not scale accordingly due to the accompanying bandwidth requirement, and the energy consumption remains high as data movement can be more expensive than computation [11–13]. In order to achieve energy-efficient CNN processing without compromising throughput, we need to develop dataflows that support parallel processing with minimal data movement. The differences in data movement energy cost based on where the data is stored also needs to be accounted for. For instance, fetching data from off-chip DRAMs costs orders of magnitude more energy than from on-chip storage [11, 12].

Many previous papers have proposed specialized CNN dataflows on various platforms, including GPU [14], FPGA [15–21], and ASIC [22–26]. However, due to differences in technology, hardware resources and system setup, a direct comparison between different implementations does not provide much insight into the relative energy efficiency of different dataflows. In this paper, we evaluate the energy efficiency of various CNN dataflows on a spatial architecture *under the same hardware resource constraints*, i.e., area, processing parallelism and technology. Based on this evaluation, we will propose *a novel dataflow that maximizes energy efficiency* for CNN acceleration.

To evaluate energy consumption, we categorize the data movements in a spatial architecture into several levels of hierarchy according to their energy cost, and then analyze each dataflow to assess the data movement at each level. This analysis framework provides insights into how each dataflow exploits different types of data movement using various architecture resources. It also offers a quantifiable way to examine the differences in energy efficiency between different dataflows.

Previously proposed dataflows typically optimize a certain type of data movement, such as input data reuse or partial

14.5 Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks

Yu-Hsin Chen[1], Tushar Krishna[1], Joel Emer[1,2], Vivienne Sze[1]

[1]Massachusetts Institute of Technology, Cambridge, MA,
[2]Nvidia, Westford, MA

Deep learning using convolutional neural networks (CNN) gives state-of-the-art accuracy on many computer vision tasks (e.g. object detection, recognition, segmentation). Convolutions account for over 90% of the processing in CNNs for both inference/testing and training, and fully convolutional networks are increasingly being used. To achieve state-of-the-art accuracy requires CNNs with not only a larger number of layers, but also millions of filters weights, and varying shapes (i.e. filter sizes, number of filters, number of channels) as shown in Fig. 14.5.1. For instance, AlexNet [1] uses 2.3 million weights (4.6MB of storage) and requires 666 million MACs per 227×227 image (13kMACs/pixel). VGG16 [2] uses 14.7 million weights (29.4MB of storage) and requires 15.3 billion MACs per 224×224 image (306kMACs/pixel). The large number of filter weights and channels results in substantial data movement, which consumes significant energy.

Existing accelerators do not support the configurability necessary to efficiently support large CNNs with different shapes [3], and using mobile GPUs can be expensive [4]. This paper describes an accelerator that can deliver state-of-the-art accuracy with minimum energy consumption in the system (including DRAM) in real-time, by using two key methods: (1) efficient dataflow and supporting hardware (spatial array, memory hierarchy and on-chip network) that minimize data movement by exploiting data reuse and support different shapes; (2) exploit data statistics to minimize energy through zeros skipping/gating to avoid unnecessary reads and computations; and data compression to reduce off-chip memory bandwidth, which is the most expensive data movement.

Figure 14.5.2 shows the top-level architecture and memory hierarchy of the accelerator. Data movement is optimized by buffering input image data (Img), filter weights (Filt) and partial sums (Psum) in a shared 108KB SRAM buffer, which facilitates temporal reuse of loaded data. Image data and filter weights are read from DRAM to the buffer and streamed into the spatial computation array allowing for overlap of memory traffic and computation. The streaming and reuse allows the system to achieve high computational efficiency even when running the memory link at a lower clock frequency than the spatial array. The spatial array computes inner products between the image and filter weights, generating partial sums that are returned from the array to the buffer and then, optionally rectified (ReLU) and compressed, to the DRAM. Run-length-based compression reduces the average image bandwidth by 2×. Configurable support for image and filter sizes that do not fit completely into the spatial array is achieved by saving partial sums in the buffer and later restoring them to the spatial array. The sizes of the spatial array and buffer determine the number of such 'passes' needed to do the calculations for a specific layer. Unused PEs are clock gated.

Figure 14.5.3 shows the dataflow within the array for filter weights, image values and partial sums. If the filter height (R) equals the number of rows in the array (in our case 12), the logical dataflow would be as follows: (1) filter weights are fed from the buffer into the left column of the array (one filter row per PE) and the filter weights move from left to right within the array; (2) image values are fed into the left column and bottom row of the array (one image row per PE) and the image values move up diagonally; (3) partial sums for each output row move up vertically, and can be read out of the top row at the end of the computational pass. If the partial sums are used in the next pass, they are fed into the bottom row of the array from the buffer at the beginning of the next computational pass.

In order to maximize utilization of a fixed-size array for different shapes, the mapping may require either folding or replication if the shape size is larger or smaller than the array dimension, respectively. Replication results in increased throughput as compared to the purely logical dataflow described above. Cases II, III, IV, and V in Fig. 14.5.3 illustrate the replication and folding of image values for various layers of AlexNet. The same data values are shown in the same color. Across the six example cases, which include physical mapping of filter weights, image values and partial sums onto the fixed-size spatial array, we see the logical dataflow patterns translating to myriad physical dataflow patterns that need to be supported. Furthermore, the same data value is often needed by multiple PEs, whose physical location in the array depends on the data type (filter, image or partial sum) and layer.

Since different layers have different shapes and hence different mappings, a design-time fixed interconnect topology will not work. Every PE can potentially be a destination for a piece of data in some particular configuration, and so a Network-on-Chip (NoC) is needed to support address based data delivery. However, traditional NoC designs with switches at every PE to buffer/forward data to one or multiple targets would result in multi-cycle delays. A full-chip broadcast to every PE could work, but would consume enormous power.

To optimize data movement, it is important to exploit spatial reuse, where a single buffer read can be used by multiple PEs (i.e. multicast). Fig. 14.5.4 shows our NoC that supports configurable data patterns, and provides an energy-efficient multicast to a variable number of PEs within a single-cycle. The NoC comprises one Global Y bus, and 12 Global X buses (one per row). Each PE is *configured* with a (row, col) ID at the beginning of processing via a scan chain. Multicast to any subset of PEs is achieved by assigning the *same* ID to *multiple* PEs. Data from the buffer is tagged with the target PEs' (row, col) ID, and multicast controllers at the input of each X bus and each PE deliver data only to those X buses and PEs, respectively, that match the target ID to avoid unnecessary switching. Data is sent on the buses only if *all* target PEs are ready (i.e., have an empty buffer) to receive. To support high bandwidth, we use separate input NoCs for filter, image, and partial sums. The partial sum NoC has a separate set of output links to the buffer to write the final partial sums. The NoC data delivery for four of the cases from Fig. 14.5.3 is shown in Fig. 14.5.4.

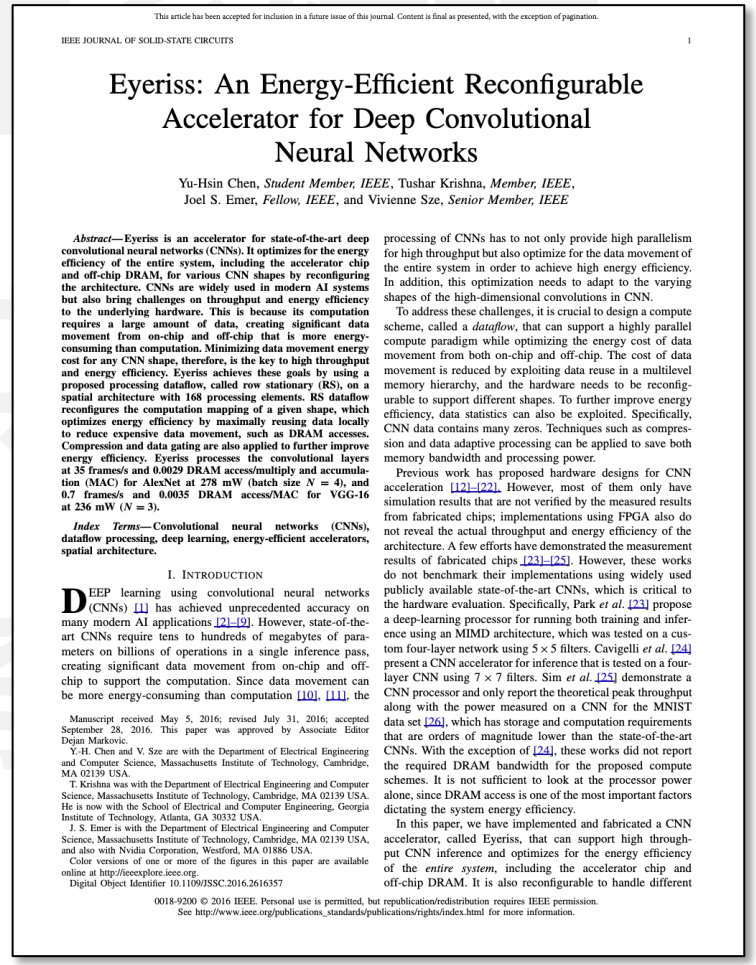
Each processing engine, shown in Fig. 14.5.5, is a three-stage pipeline responsible for calculating the inner product of the input image and filter weights for a single row of the filter. The sequence of partial sums for the sliding filter window is computed sequentially. The partial sums for the row are passed on a local link to the neighboring PE (see Fig. 14.5.4), where the cross-row partial sums are computed. Local scratch pads allow for energy-efficient temporal reuse of input image and filter weights by recirculating values needed by different windows. A partial sum scratch pad allows for temporal reuse of partial sums being generated for different images and/or channels and filters. Data gating is achieved by recording the input image values of zero in a 'zero buffer' and skipping filter reads and computation for those values resulting in a 45% power savings in the PE.

The test chip is implemented in 65nm CMOS. It operates at 200MHz core clock and 60MHz link clock, which results in a frame rate of 34.7fps on the five convolutional layers in AlexNet and a measured power of 278mW at 1V. The PE array, NoC and on-chip buffer consume 77.8%, 15.6% and 2.7% of the total power, respectively. The core and link clocks can scale up to 250MHz and 90MHz, respectively. This enables us to achieve a throughput of 44.8fps at 1.17V. Fig. 14.5.6 shows the performance at each layer, including compression ratio, power consumption, PE utilization, and memory access to highlight the reduction in DRAM bandwidth, efficiency of the reconfigurable mapping and reduced data access due to data reuse, respectively. A die photo of the chip and the range of the shapes it can support natively are shown in Fig. 14.5.7.

Acknowledgements:
This work is funded by the DARPA YFA grant N66001-14-1-4039, MIT Center for Integrated Circuits & Systems, and a gift from Intel. The authors would also like to thank Mehul Tikekar and Michael Price for their technical assistance.

References:

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Neural Information Processing Systems*, pp. 1097–1105, 2012.
[2] K. Simonyan, A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," CoRR, abs/1409.1556, 2014.
[3] S. Park et al., "A 1.93TOPS/W Scalable Deep Learning/Inference Processor with Tetra-parallel MIMD Architecture for Big Data Applications," *ISSCC Dig. Tech. Papers*, pp. 80-81, 2015
[4] S. Chetlur et al., "cuDNN: Efficient Primitives for Deep Learning," CoRR, abs/1410.0759, 2014.



This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

IEEE JOURNAL OF SOLID-STATE CIRCUITS 1

Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks

Yu-Hsin Chen, *Student Member, IEEE*, Tushar Krishna, *Member, IEEE*, Joel S. Emer, *Fellow, IEEE*, and Vivienne Sze, *Senior Member, IEEE*

***Abstract*— Eyeriss is an accelerator for state-of-the-art deep convolutional neural networks (CNNs). It optimizes for the energy efficiency of the entire system, including the accelerator chip and off-chip DRAM, for various CNN shapes by reconfiguring the architecture. CNNs are widely used in modern AI systems but also bring challenges on throughput and energy efficiency to the underlying hardware. This is because its computation requires a large amount of data, creating significant data movement from on-chip and off-chip that is more energy-consuming than computation. Minimizing data movement energy cost for any CNN shape, therefore, is the key to high throughput and energy efficiency. Eyeriss achieves these goals by using a proposed processing dataflow, called row stationary (RS), on a spatial architecture with 168 processing elements. RS dataflow reconfigures the computation mapping of a given shape, which optimizes energy efficiency by maximally reusing data locally to reduce expensive data movement, such as DRAM accesses. Compression and data gating are also applied to further improve energy efficiency. Eyeriss processes the convolutional layers at 35 frames/s and 0.0029 DRAM access/multiply and accumulation (MAC) for AlexNet at 278 mW (batch size $N = 4$), and 0.7 frames/s and 0.0035 DRAM access/MAC for VGG-16 at 236 mW ($N = 3$).**

***Index Terms*— Convolutional neural networks (CNNs), dataflow processing, deep learning, energy-efficient accelerators, spatial architecture.**

I. Introduction

DEEP learning using convolutional neural networks (CNNs) [1] has achieved unprecedented accuracy on many modern AI applications [2]–[9]. However, state-of-the-art CNNs require tens to hundreds of megabytes of parameters on billions of operations in a single inference pass, creating significant data movement from on-chip and off-chip to support the computation. Since data movement can be more energy-consuming than computation [10], [11], the processing of CNNs has to not only provide high parallelism for high throughput but also optimize for the data movement of the entire system in order to achieve high energy efficiency. In addition, this optimization needs to adapt to the varying shapes of the high-dimensional convolutions in CNN.

To address these challenges, it is crucial to design a compute scheme, called a *dataflow*, that can support a highly parallel compute paradigm while optimizing the energy cost of data movement from both on-chip and off-chip. The cost of data movement is reduced by exploiting data reuse in a multilevel memory hierarchy, and the hardware needs to be reconfigurable to support different shapes. To further improve energy efficiency, data statistics can also be exploited. Specifically, CNN data contains many zeros. Techniques such as compression and data adaptive processing can be applied to save both memory bandwidth and processing power.

Previous work has proposed hardware designs for CNN acceleration [12]–[22]. However, most of them only have simulation results that are not verified by the measured results from fabricated chips; implementations using FPGA also do not reveal the actual throughput and energy efficiency of the architecture. A few efforts have demonstrated the measurement results of fabricated chips [23]–[25]. However, these works do not benchmark their implementations using widely used publicly available state-of-the-art CNNs, which is critical to the hardware evaluation. Specifically, Park *et al.* [23] propose a deep-learning processor for running both training and inference using an MIMD architecture, which was tested on a custom four-layer network using 5 × 5 filters. Cavigelli *et al.* [24] present a CNN accelerator for inference that is tested on a four-layer CNN using 7 × 7 filters. Sim *et al.* [25] demonstrate a CNN processor and only report the theoretical peak throughput along with the power measured on a CNN for the MNIST data set [26], which has storage and computation requirements that are orders of magnitude lower than the state-of-the-art CNNs. With the exception of [24], these works did not report the required DRAM bandwidth for the proposed compute schemes. It is not sufficient to look at the processor power alone, since DRAM access is one of the most important factors dictating the system energy efficiency.

In this paper, we have implemented and fabricated a CNN accelerator, called Eyeriss, that can support high throughput CNN inference and optimizes for the energy efficiency of the *entire system*, including the accelerator chip and off-chip DRAM. It is also reconfigurable to handle different

Manuscript received May 5, 2016; revised July 31, 2016; accepted September 28, 2016. This paper was approved by Associate Editor Dejan Markovic.

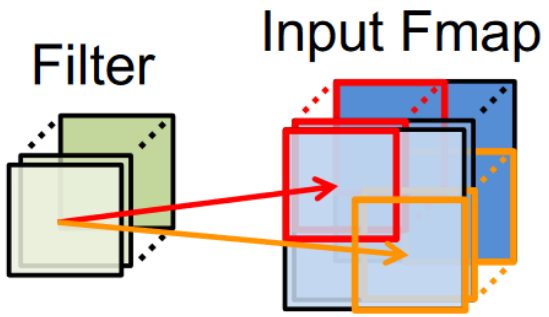
Y.-H. Chen and V. Sze are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.
T. Krishna was with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA. He is now with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA.
J. S. Emer is with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA, and also with Nvidia Corporation, Westford, MA 01886 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.
Digital Object Identifier 10.1109/JSSC.2016.2616357

AI加速器架构发展——Eyeriss

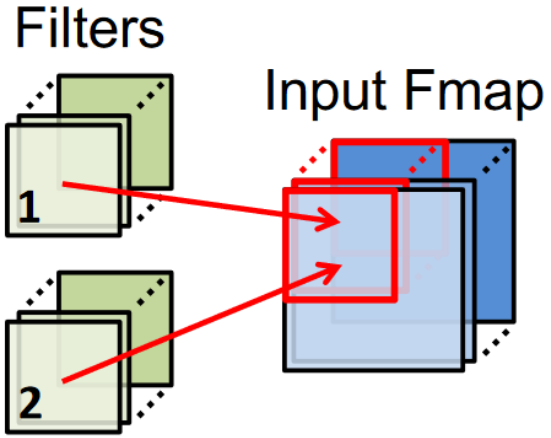
- 卷积计算中的**数据复用模式**
  - 通过权重和激活值的复用，可以显著降低访存需求，提升计算效率

卷积层



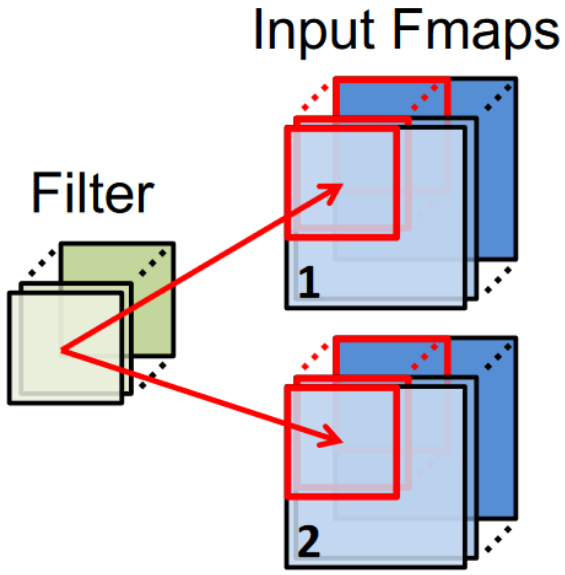
激活值和权重值的复用

卷积层和全连接层



激活值的复用

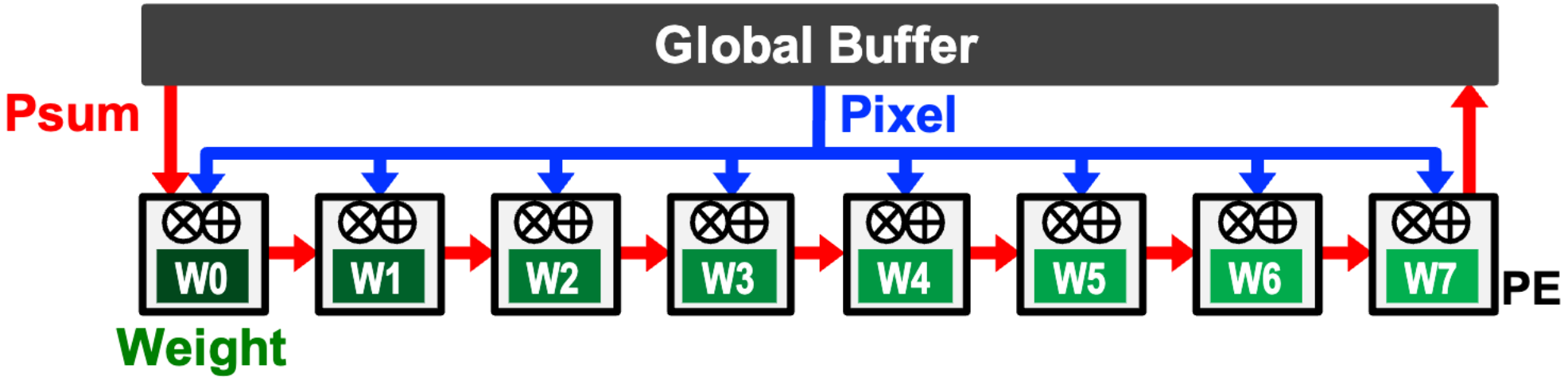
卷积层和全连接层



权重值的复用

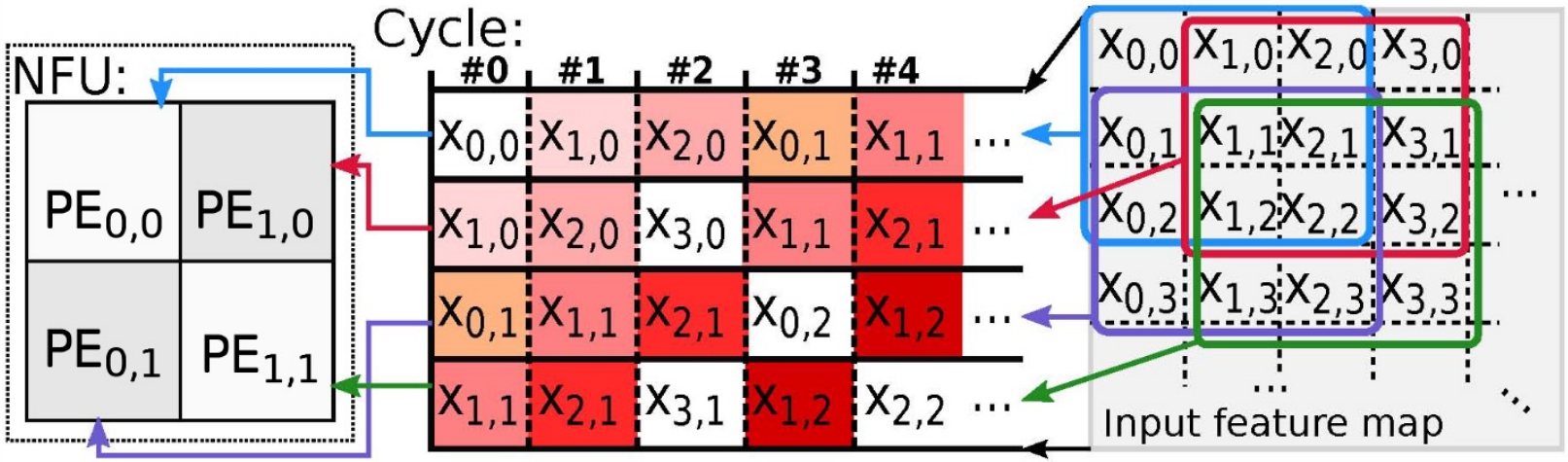
AI加速器架构发展——Eyeriss

- Eyeriss把现有数据流加速器分为3类：
  - **权重静态**，代表工作包括TPU等，运算过程中，权重值被预先存储在PE中，最大化权重复用



AI加速器架构发展——Eyeriss

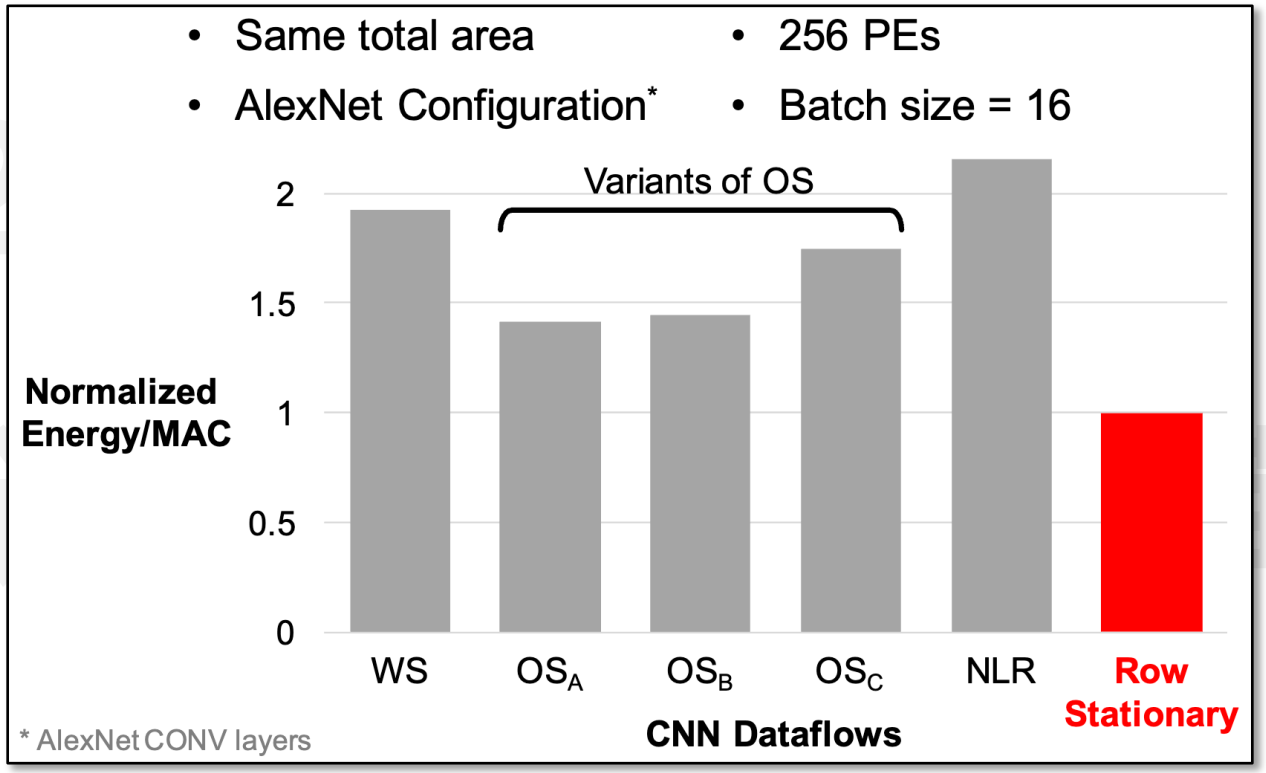
- Eyeriss把现有数据流加速器分为3类：
  - **权重静态**，代表工作包括TPU等，权重值被预先存储在PE中，最大化权重复用
  - **输出静态**，代表工作包括ShiDianNao等，输出值停留在PE中被累加，最小化输出的读写开销



ShiDianNao

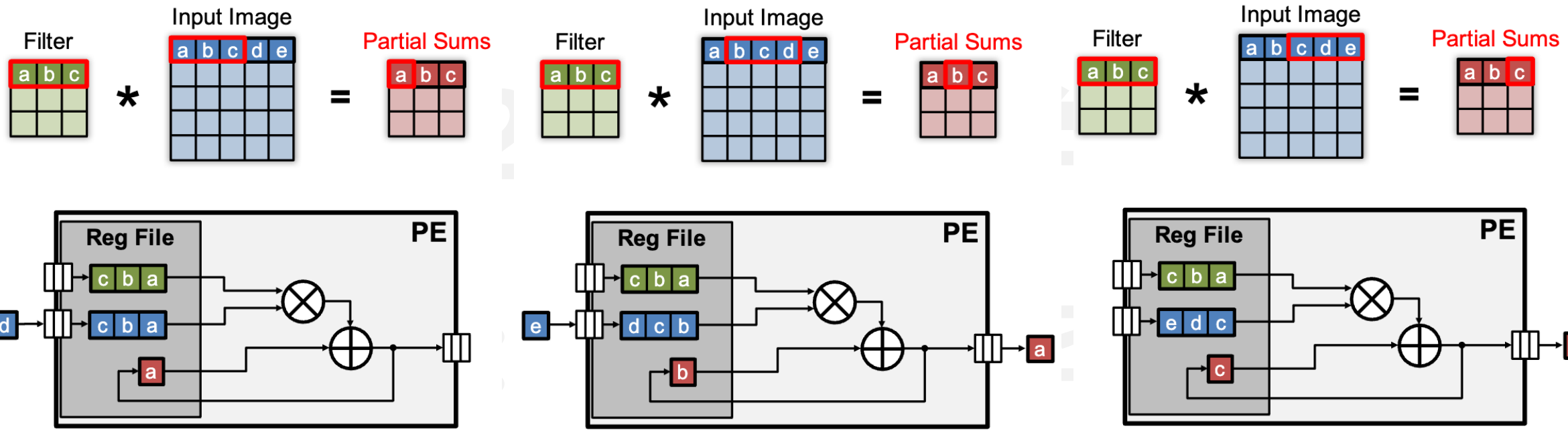
AI加速器架构发展——Eyeriss

- Eyeriss把现有数据流加速器分为3类：
  - **权重静态**，代表工作包括TPU等，权重值被预先存储在PE中，最大化权重复用
  - **输出静态**，代表工作包括ShiDianNao等，输出值停留在PE中被累加，最小化输出的读写开销
  - **No local reuse**，代表工作包括DianNao、DaDianNao等



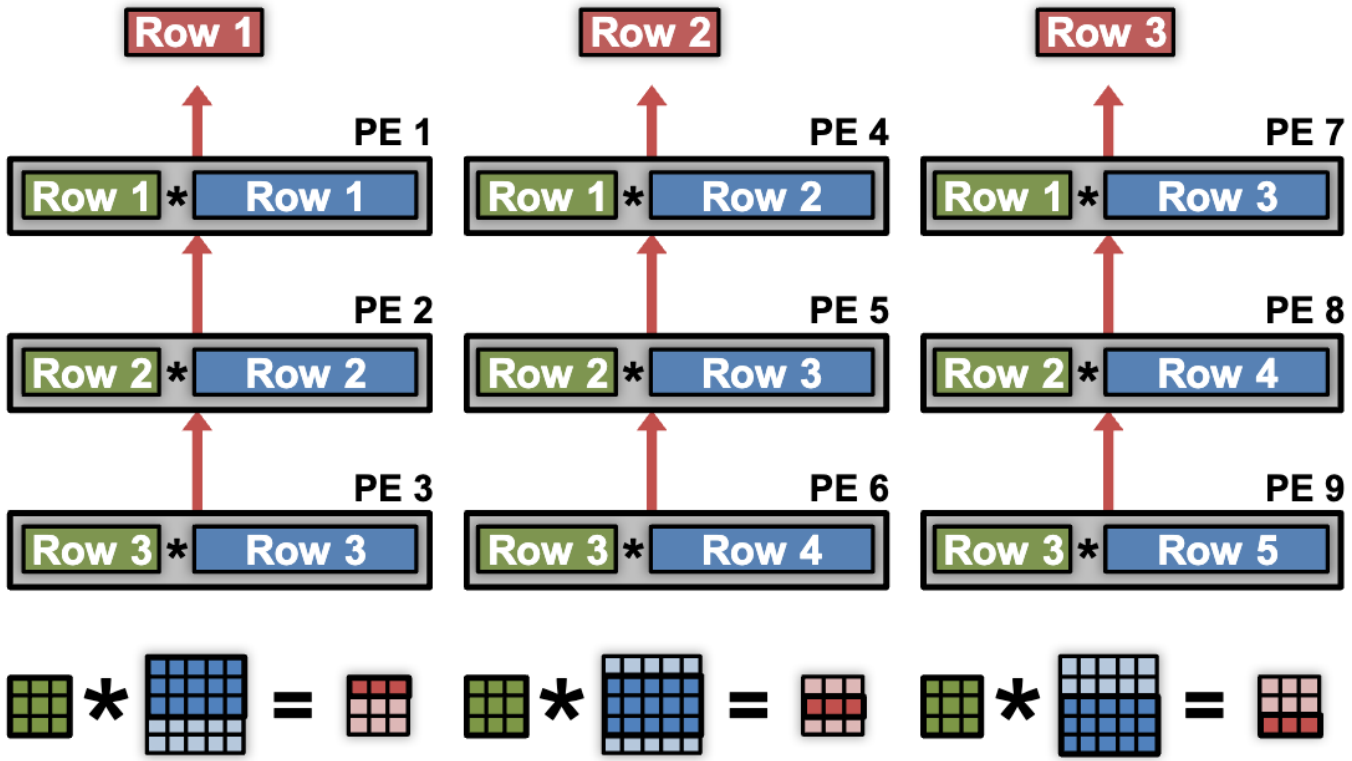
AI加速器架构发展——Eyeriss

- 基于数据流分类，Eyeriss提出了row stationary的数据流
  - 计算单元内部，权重、输入和输出激活值均被复用



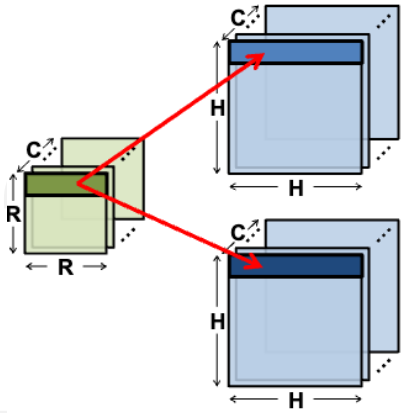
AI加速器架构发展——Eyeriss

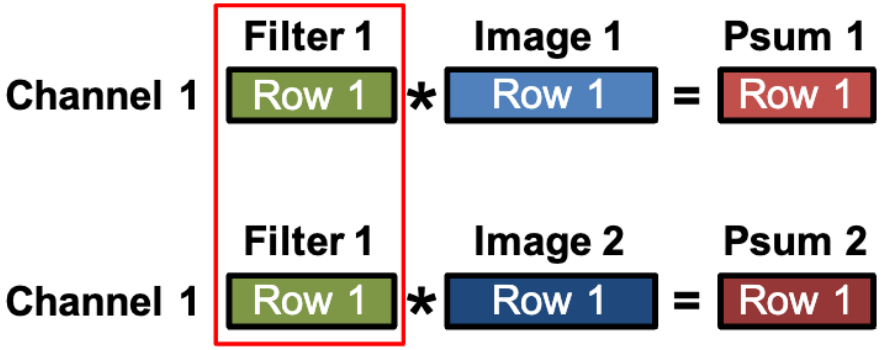
- 基于数据流分类，Eyeriss提出了row stationary的数据流
  - 计算单元内部，权重、输入和输出激活值均被复用
  - 不同计算单元间，进一步复用权重、输入和输出



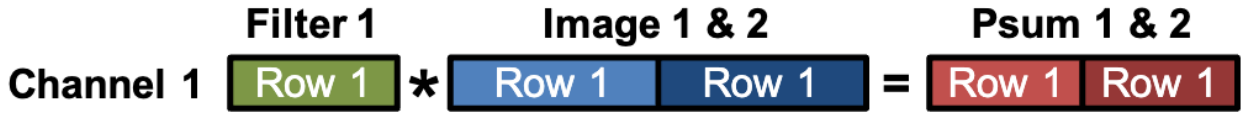
AI加速器架构发展——Eyeriss

- 基于数据流分类，Eyeriss提出了row stationary的数据流
  - 计算单元内部，权重、输入和输出激活值均被复用
  - 不同计算单元间，进一步复用权重、输入和输出
  - 进一步拓展到多输入图像



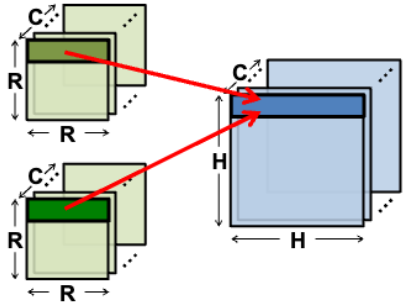


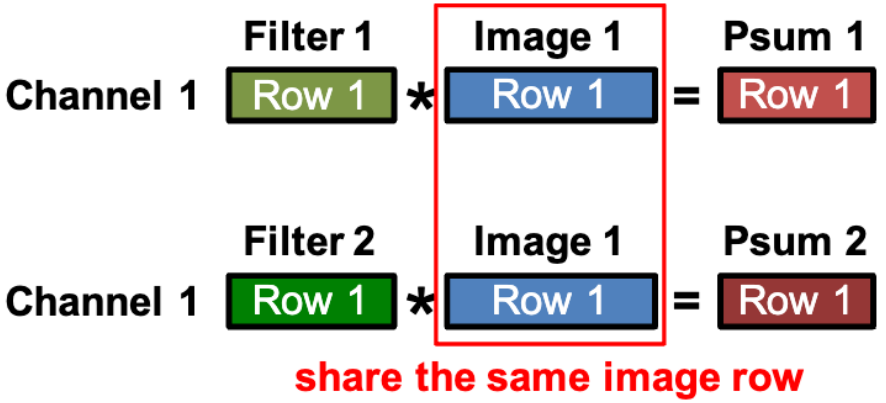
Processing in PE: **concatenate image rows**



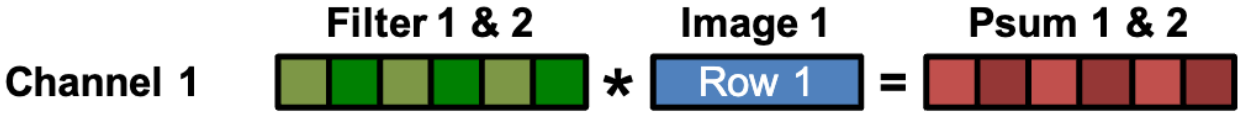
AI加速器架构发展——Eyeriss

- 基于数据流分类，Eyeriss提出了row stationary的数据流
  - 计算单元内部，权重、输入和输出激活值均被复用
  - 不同计算单元间，进一步复用权重、输入和输出
  - 进一步拓展到多输入图像、多卷积核



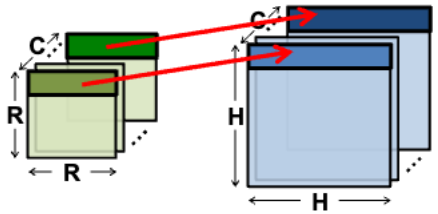


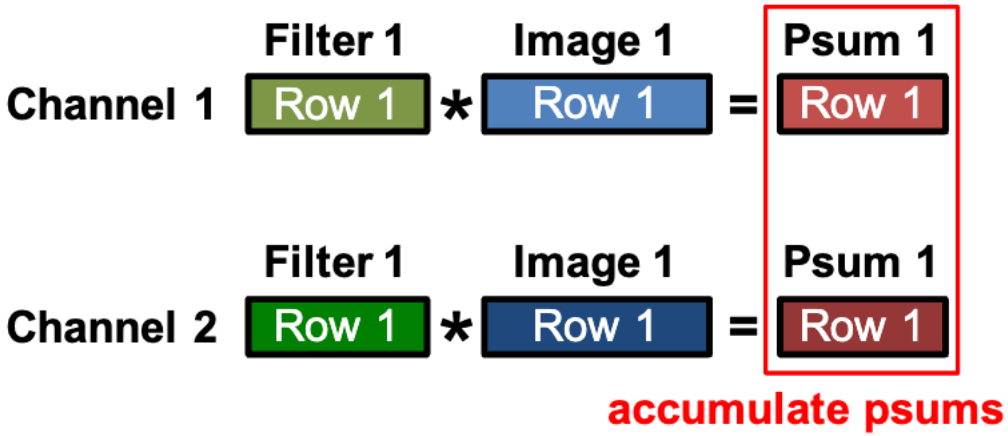
Processing in PE: **interleave filter rows**



AI加速器架构发展——Eyeriss

- 基于数据流分类，Eyeriss提出了row stationary的数据流
  - 计算单元内部，权重、输入和输出激活值均被复用
  - 不同计算单元间，进一步复用权重、输入和输出
  - 进一步拓展到多输入图像、多卷积核，不同输入通道

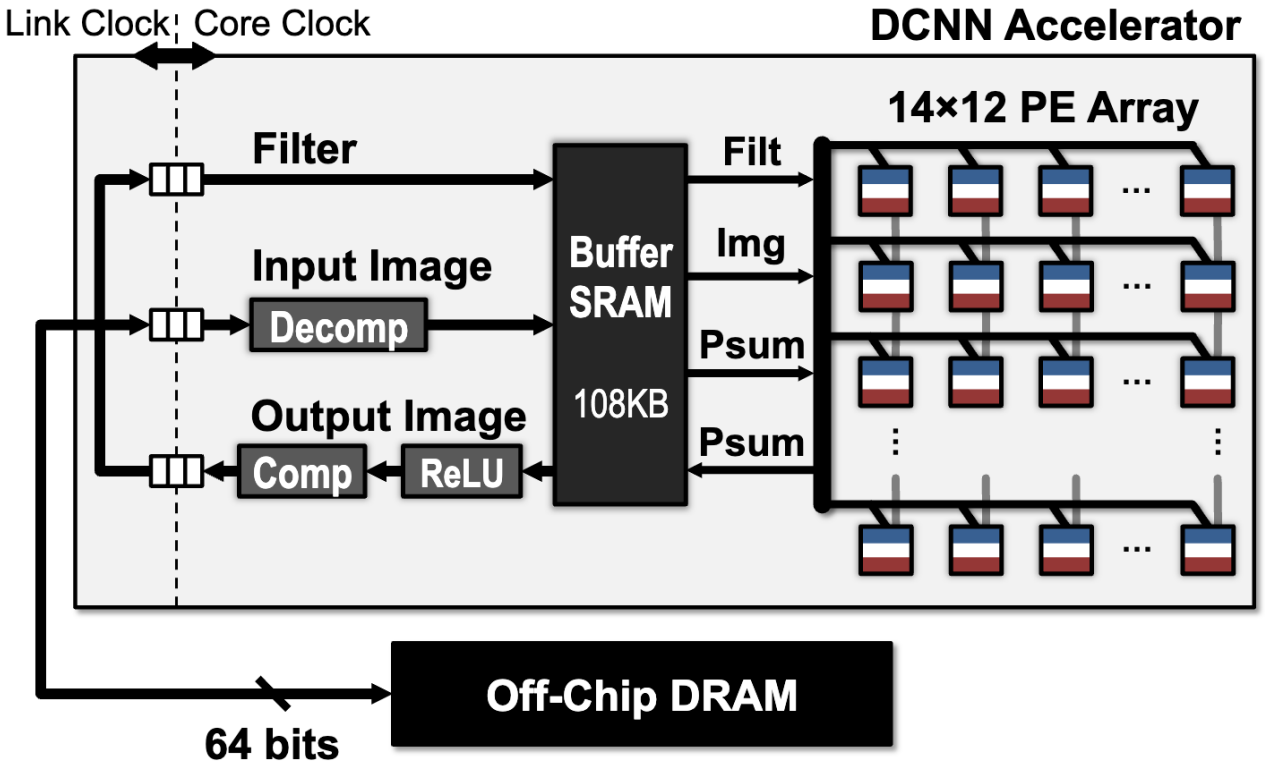




Processing in PE: **interleave channels**

AI加速器架构发展——Eyeriss

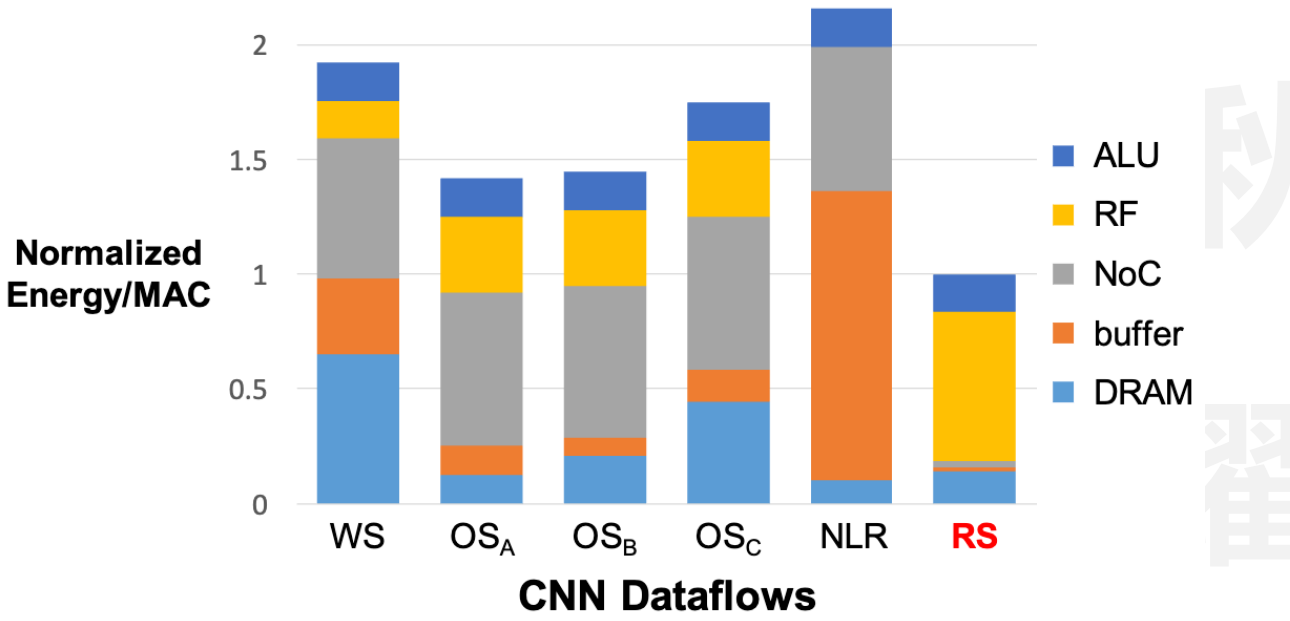
- Eyeriss的存储系统
  - 计算单元内部需要交大的便签存储器，提升数据复用
  - 采用了统一的便签存储器，**容量很小**，主要用于计算卷积层
  - 插入了压缩和解压缩模块，能够利用输入和输出结果中的0元素

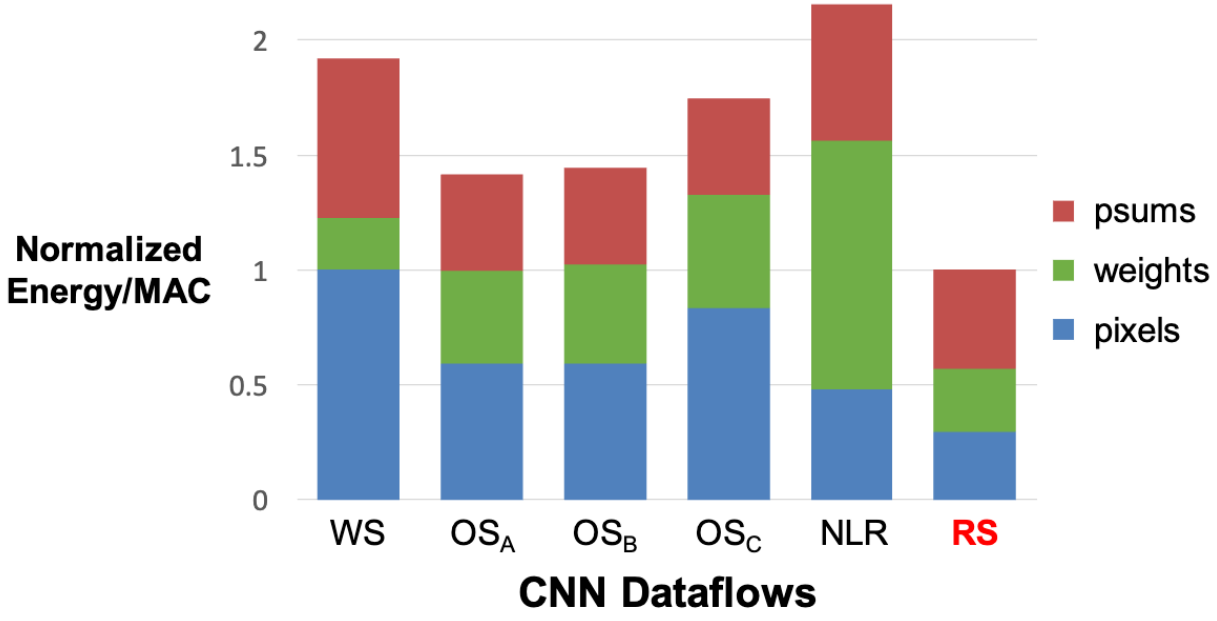


Technology	TSMC 65nm LP 1P9M
Core Area	3.5mm×3.5mm
Gate Count	1852 kGates (NAND2)
On-Chip Buffer	108 KB
# of PEs	168
Scratch Pad / PE	0.5 KB
Supply Voltage	0.82 – 1.17 V
Core Frequency	100 – 250 MHz
Peak Performance	33.6 – 84.0 GOPS (2 OP = 1 MAC)
Word Bit-width	16-bit Fixed-Point
Filter Size*	1 – 32 [width] 1 – 12 [height]
# of Filters*	1 – 1024
# of Channels*	1 – 1024
Stride Range	1–12 [horizontal] 1, 2, 4 [vertical]

AI加速器架构发展——Eyeriss

- 通过将Eyeriss的功耗与其他数据流进行比较可以发现
  - RF的功耗显著提升（每位每个PE中的存储显著提升）
  - Buffer的功耗很小，因为进一步提升了输入、权重和输出数据的复用





AI加速器架构发展——Google TPU v1

北京大学 PEKING UNIVERSITY

- Google TPU论文发表于ISCA 2016，从2015年开始，TPU已经集成在了Google服务器中
- 只考虑神经网络推理加速，重点关注MLP、CNN、LSTM三类神经网络
- 相较于NV K80 GPU和Intel Haswell CPU, 15-30倍延迟降低、30-80倍能效降低

Name	*Layers*					*Nonlinear function*	*Weights*	*TPU Ops / Weight Byte*	*TPU Batch Size*	*% of Deployed TPUs in July 2016*
	FC	*Conv*	*Vector*	*Pool*	*Total*					
MLP0	5				5	ReLU	20M	200	200	61%
MLP1	4				4	ReLU	5M	168	168	
LSTM0	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	37		19		56	sigmoid, tanh	34M	96	96	
CNN0		16			16	ReLU	8M	2888	8	5%
CNN1	4	72		13	89	ReLU	100M	1750	32	

In-Datacenter Performance Analysis of a Tensor Processing Unit™

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon

Google, Inc., Mountain View, CA USA

Email: {jouppi, cliffy, nishantpatil, davidpatterson}@google.com

To appear at the 44th International Symposium on Computer Architecture (ISCA), Toronto, Canada, June 26, 2017.

Abstract

Many architects believe that major improvements in cost-energy-performance must now come from domain-specific hardware. This paper evaluates a custom ASIC—called a *Tensor Processing Unit (TPU)*— deployed in datacenters since 2015 that accelerates the inference phase of neural networks (NN). The heart of the TPU is a 65,536 8-bit MAC matrix multiply unit that offers a peak throughput of 92 TeraOps/second (TOPS) and a large (28 MiB) software-managed on-chip memory. The TPU's deterministic execution model is a better match to the 99th-percentile response-time requirement of our NN applications than are the time-varying optimizations of CPUs and GPUs (caches, out-of-order execution, multithreading, multiprocessing, prefetching, …) that help average throughput more than guaranteed latency. The lack of such features helps explain why, despite having myriad MACs and a big memory, the TPU is relatively small and low power. We compare the TPU to a server-class Intel Haswell CPU and an Nvidia K80 GPU, which are contemporaries deployed in the same datacenters. Our workload, written in the high-level TensorFlow framework, uses production NN applications (MLPs, CNNs, and LSTMs) that represent 95% of our datacenters' NN inference demand. Despite low utilization for some applications, the TPU is on average about 15X - 30X faster than its contemporary GPU or CPU, with TOPS/Watt about 30X - 80X higher. Moreover, using the GPU's GDDR5 memory in the TPU would triple achieved TOPS and raise TOPS/Watt to nearly 70X the GPU and 200X the CPU.

Index terms–DNN, MLP, CNN, RNN, LSTM, neural network, domain-specific architecture, accelerator

1. Introduction to Neural Networks

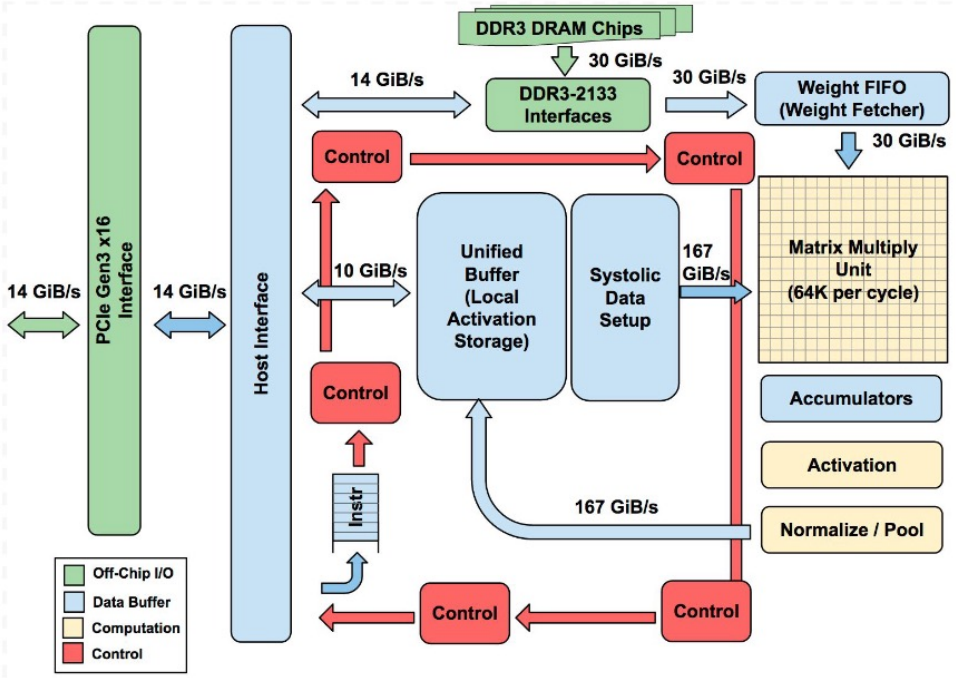
The synergy between the large data sets in the cloud and the numerous computers that power it has enabled a renaissance in machine learning. In particular, *deep neural networks* (DNNs) have led to breakthroughs such as reducing word error rates in speech recognition by 30% over traditional approaches, which was the biggest gain in 20 years [Dea16]; cutting the error rate in an image recognition competition since 2011 from 26% to 3.5% [Kri12] [Sze15] [He16]; and beating a human champion at Go [Sil16]. Unlike some hardware targets, DNNs are applicable to a wide range of problems, so we can reuse a DNN-specific ASIC for solutions in speech, vision, language, translation, search ranking, and many more.

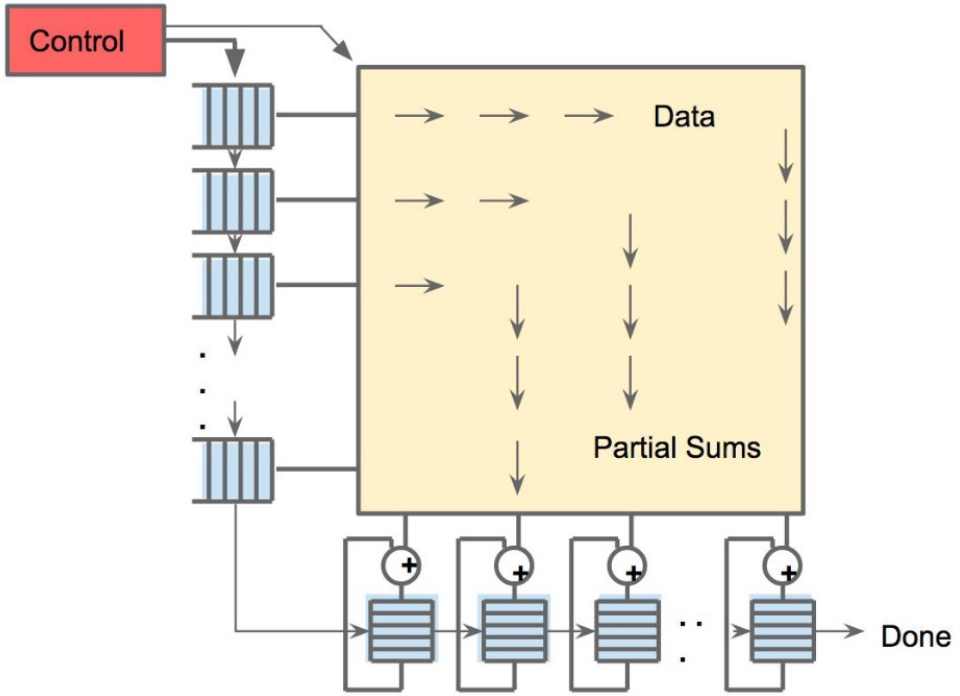
Neural networks (NN) target brain-like functionality and are based on a simple artificial neuron: a nonlinear function (such as `max(0, value)`) of a weighted sum of the inputs. These artificial neurons are collected into layers, with the outputs of one layer becoming the inputs of the next one in the sequence. The "deep" part of DNN comes from going beyond a few layers, as the large data sets in the cloud allowed more accurate models to be built by using extra and larger layers to capture higher levels of patterns or concepts, and GPUs provided enough computing to develop them.

The two phases of NN are called *training* (or learning) and *inference* (or prediction), and they refer to development versus production. The developer chooses the number of layers and the type of NN, and training determines the weights. Virtually all training today is in floating point, which is one reason GPUs have been so popular. A step called *quantization* transforms floating-point numbers into narrow integers—often just 8 bits—which are usually good enough for inference. Eight-bit integer multiplies can be 6X less energy and 6X less area than IEEE 754 16-bit floating-point multiplies, and the

AI加速器架构发展——Google TPU

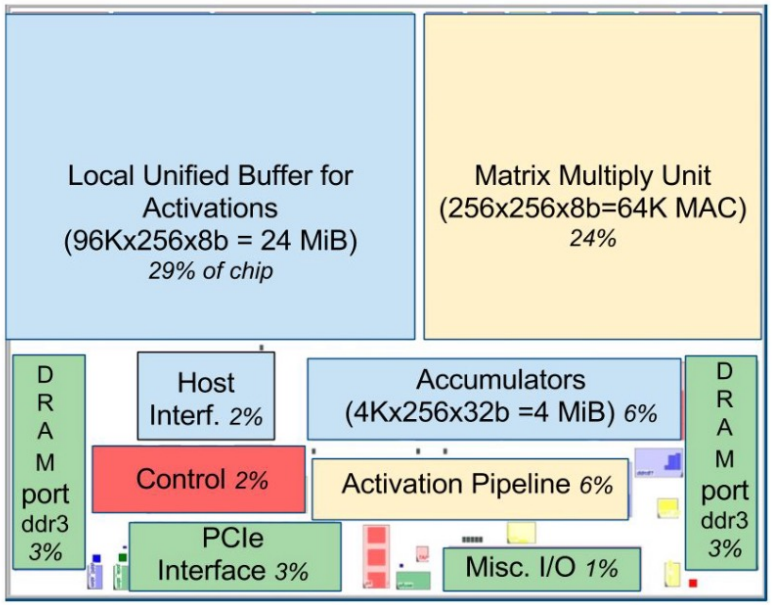
- TPU设计参数
  - 矩阵乘法单元：256 x 256，支持稠密矩阵乘法（不支持稀疏计算），double buffering，每周期产生256个partial sum，支持8比特权重，8/16比特输入
  - 累加单元：4 MB缓存（4096 x 256 x 32b），double buffering
  - 片上存储: MMU存储64KB权重，FIFO深度为4，激活存储为24MB

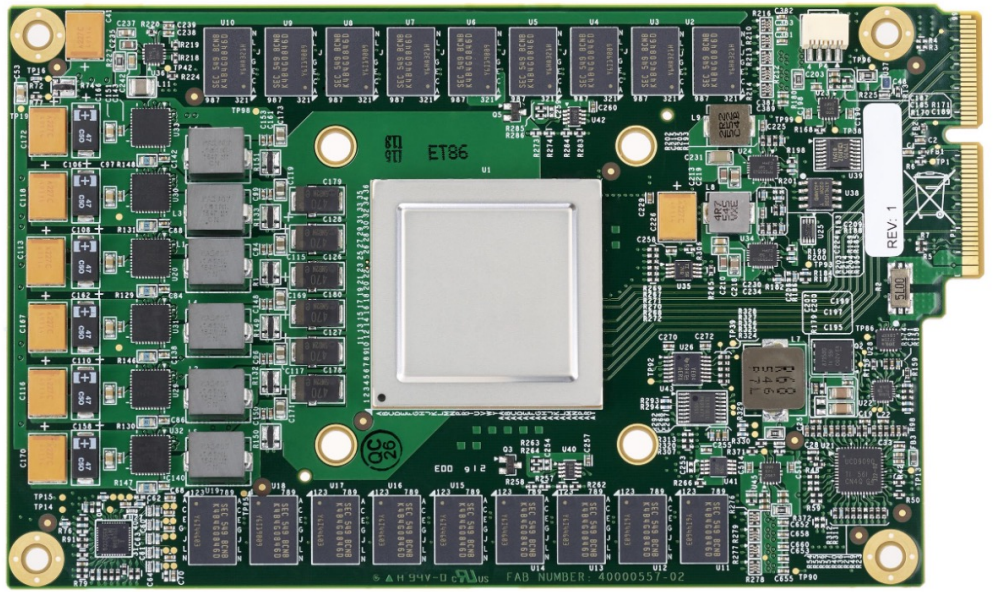




AI加速器架构发展——Google TPU

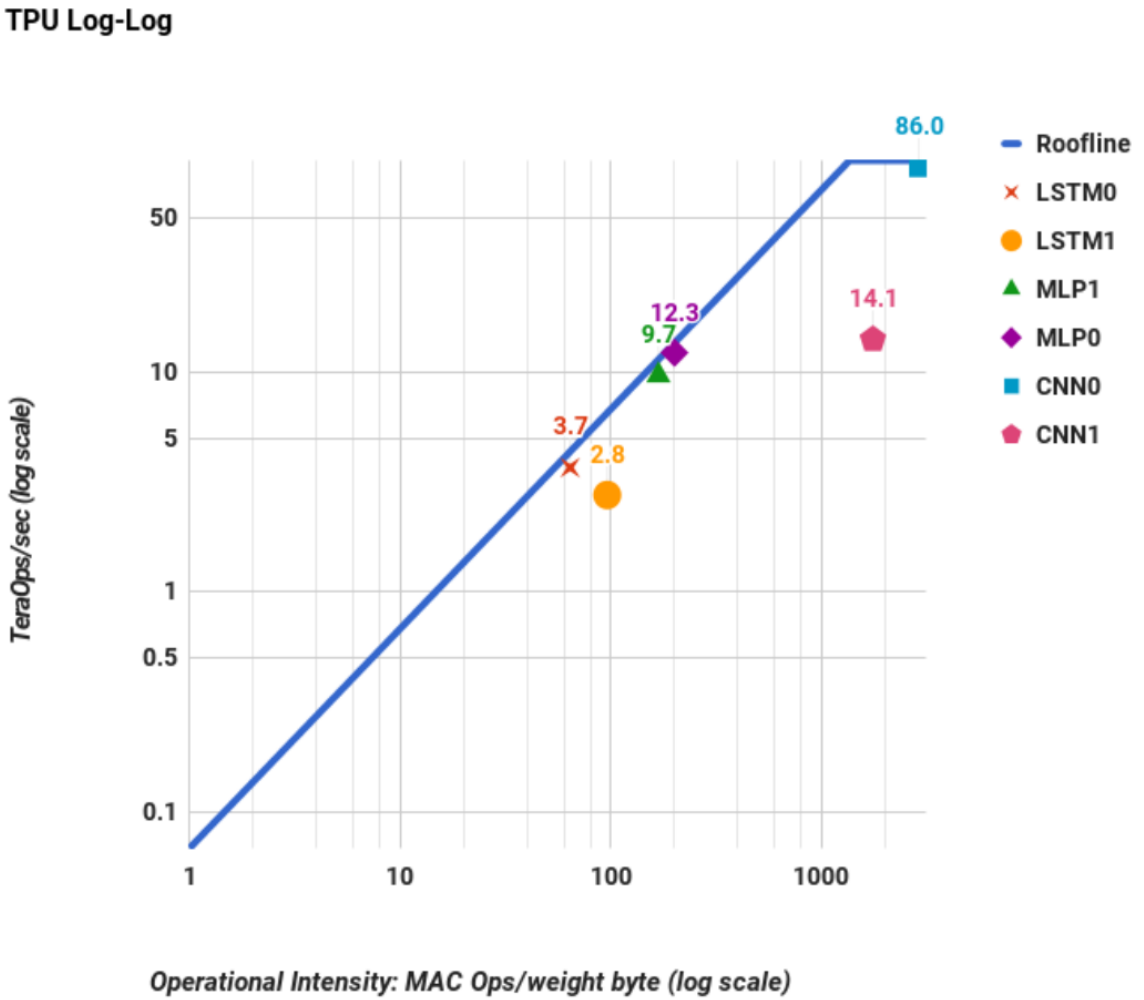
- 在TPU的版图中，数据通路面积占比为67%，I/O面积占比10%，控制占比2%
- 采用CISC指令，其中5个重要指令包括
  - Read_Host_Memory, Read_Weights, MatrixMultiply/Convolve, Activate, Write_Host_Memory





AI加速器架构发展——Google TPU

- 基于Roofline模型的系统瓶颈评估
- TPU Roofline：
  - 每Byte的权重读取需要1350乘加计算平摊，才能使得系统不会成为访存瓶颈
  - 访存瓶颈的操作：LSTM、MLP
  - 计算瓶颈的操作：卷积



AI加速器架构发展——Google TPU

北京大学 PEKING UNIVERSITY

- Google TPU v1的性能分析
  - 针对MLP、LSTM，模型权重的加载成为主要瓶颈
  - 实际算力与峰值算力之间存在显著差距

Name	Layers					Nonlinear function	Weights	TPU Ops / Weight Byte
	FC	Conv	Vector	Pool	Total			
MLP0	5				5	ReLU	20M	200
MLP1	4				4	ReLU	5M	168
LSTM0	24		34		58	sigmoid, tanh	52M	64
LSTM1	37		19		56	sigmoid, tanh	34M	96
CNN0		16			16	ReLU	8M	2888
CNN1	4	72		13	89	ReLU	100M	1750

Application	MLP0	MLP1	LSTM0	LSTM1	CNN0	CNN1	Mean	Row
Array active cycles	12.7%	10.6%	8.2%	10.5%	78.2%	46.2%	28%	1
Useful MACs in 64K matrix (% peak)	12.5%	9.4%	8.2%	6.3%	78.2%	22.5%	23%	2
Unused MACs	0.3%	1.2%	0.0%	4.2%	0.0%	23.7%	5%	3
Weight stall cycles	53.9%	44.2%	58.1%	62.1%	0.0%	28.1%	43%	4
Weight shift cycles	15.9%	13.4%	15.8%	17.1%	0.0%	7.0%	12%	5
Non-matrix cycles	17.5%	31.9%	17.9%	10.3%	21.8%	18.7%	20%	6
RAW stalls	3.3%	8.4%	14.6%	10.6%	3.5%	22.8%	11%	7
Input data stalls	6.1%	8.8%	5.1%	2.4%	3.4%	0.6%	4%	8
TeraOps/sec (92 Peak)	12.3	9.7	3.7	2.8	86.0	14.1	21.4	9

Table 3. Factors limiting TPU performance of the NN workload based on hardware performance counters. Rows 1, 4, 5, and 6 total 100% and are based on measurements of activity of the matrix unit. Rows 2 and 3 further break down the fraction of 64K weights in the matrix unit that hold useful weights on active cycles. Our counters cannot exactly explain the time when the matrix unit is idle in row 6; rows 7 and 8 show counters for two possible reasons, including RAW pipeline hazards and PCIe input stalls. Row 9 (TOPS) is based on measurements of production code while the other rows are based on performance-counter measurements, so they are not perfectly consistent. Host server overhead is excluded here. CNN1 results are explained in the text.

AI加速器架构发展——Google TPU v2

- 不同于TPU v1只支持推理，TPU v2需要同时支持训练
- 相较于推理，支持训练的难度更大：
  - 更多计算（和类型）：反向传播、转置、求导等操作
  - 更多存储：中间计算需要保存，反向传播中会用到
  - 更高精度：INT8无法满足训练需求
  - 更高的可编程性：不同的优化器等
  - 更难并行计算：

Build it quickly

Achieve high performance...

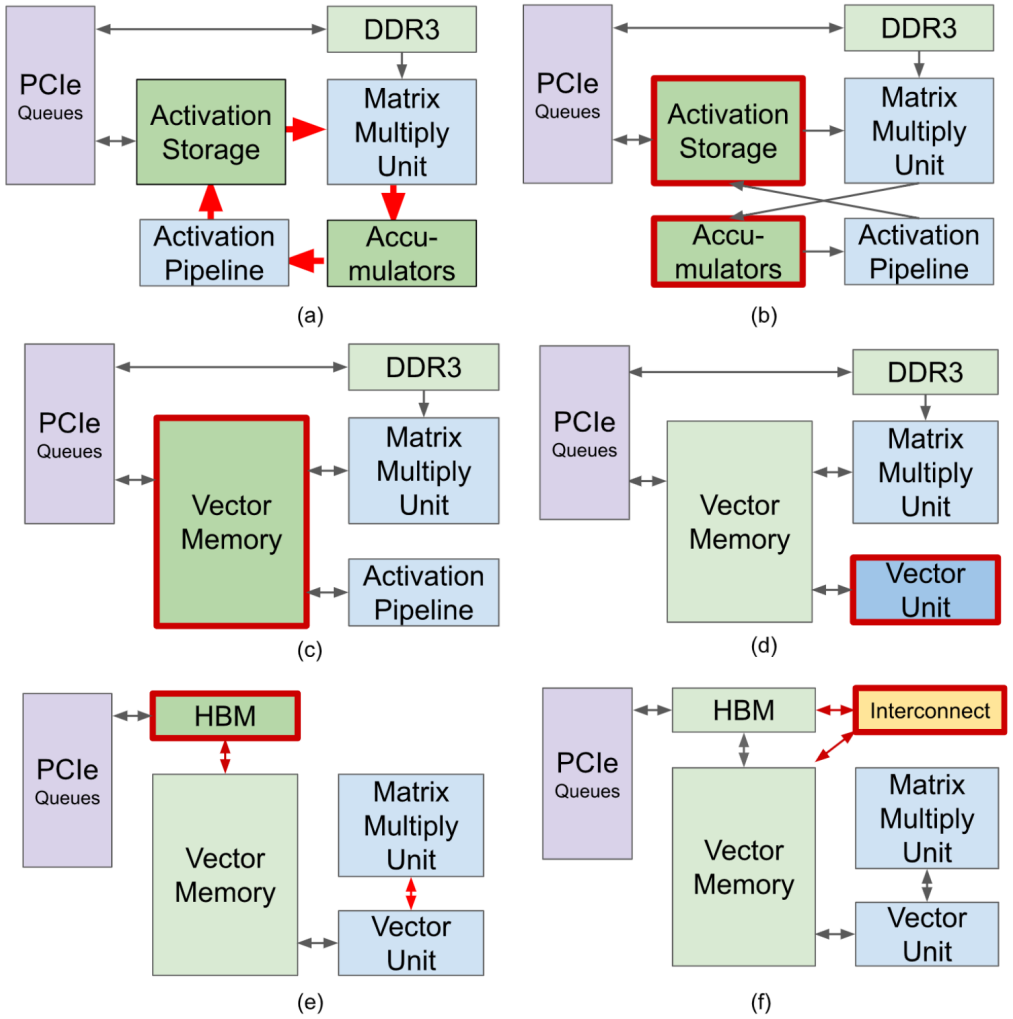
...at scale

...for new workloads out-of-the-box

...all while being cost effective

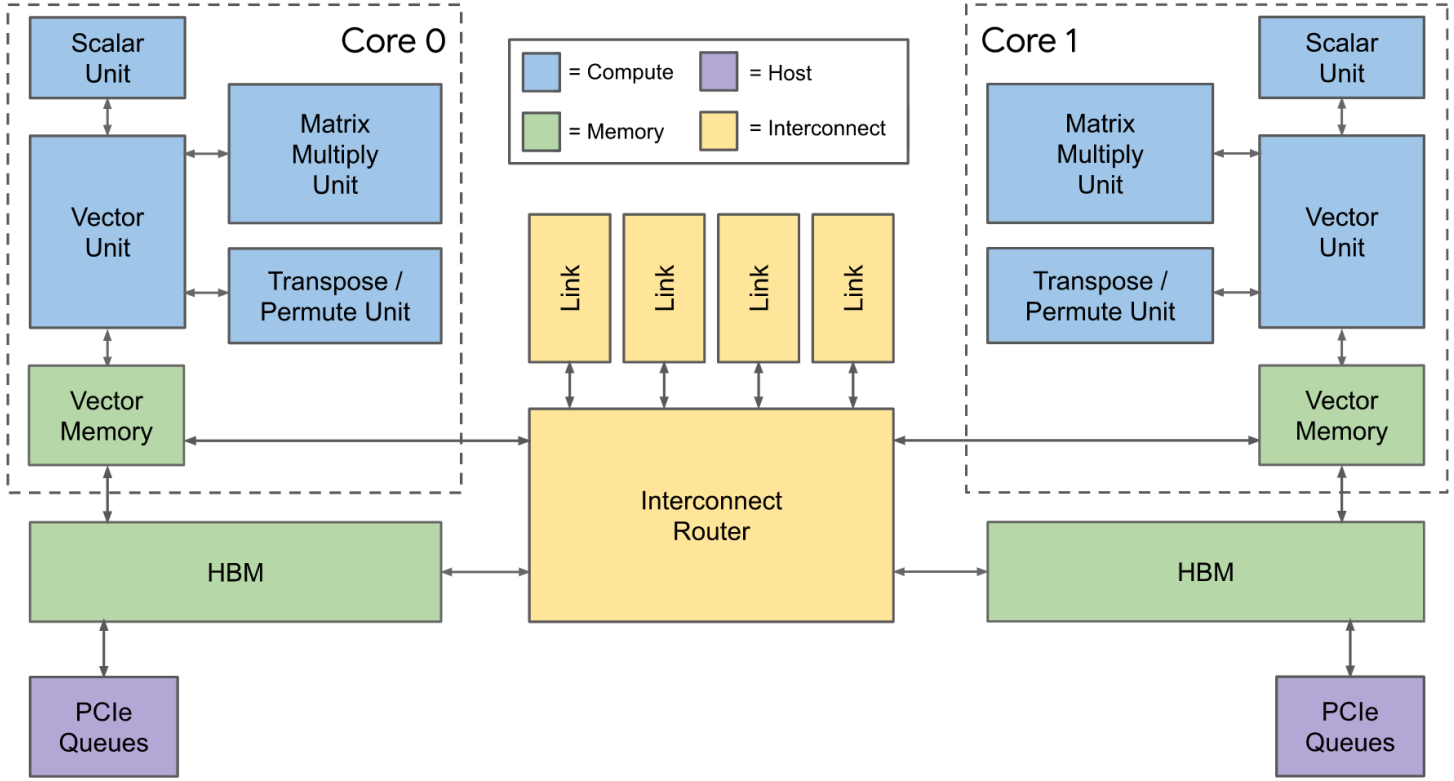
AI加速器架构发展——Google TPU v2

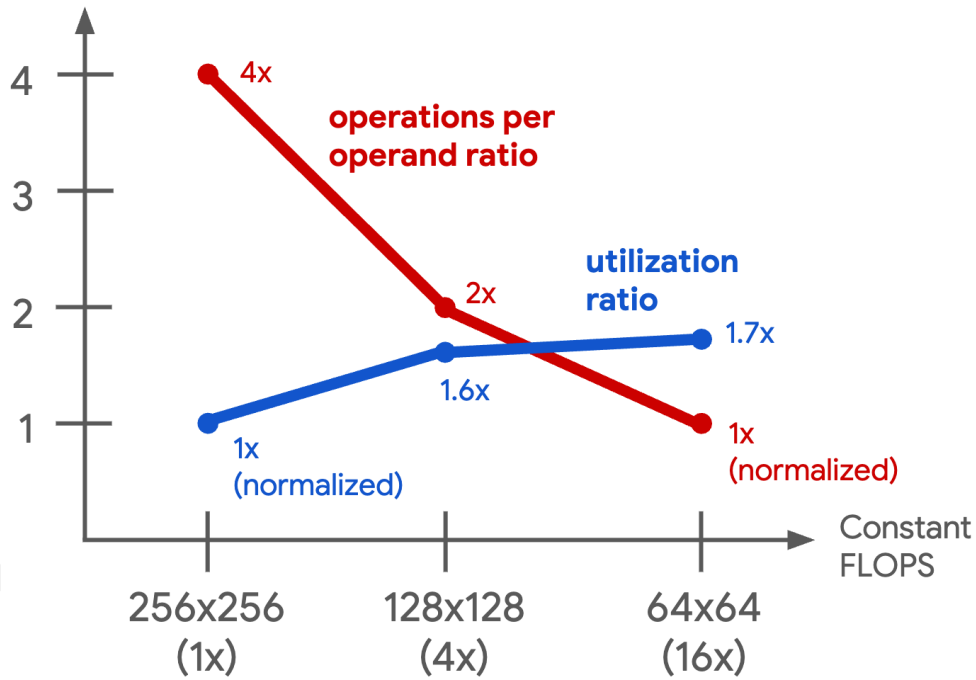
- 相较于TPU v1，TPU v2的**核心改动包括**
  - 将累加便签存储器、激活值便签存储器合并
  - 将累加器改为更为灵活、可编程的向量处理单元
  - 将MMU改为向量单元的协处理器，简化访存需求
  - 将DDR3改为HBM，提供更高访存带宽
  - 增加多卡互联，提升多卡扩展效率



AI加速器架构发展——Google TPU v2

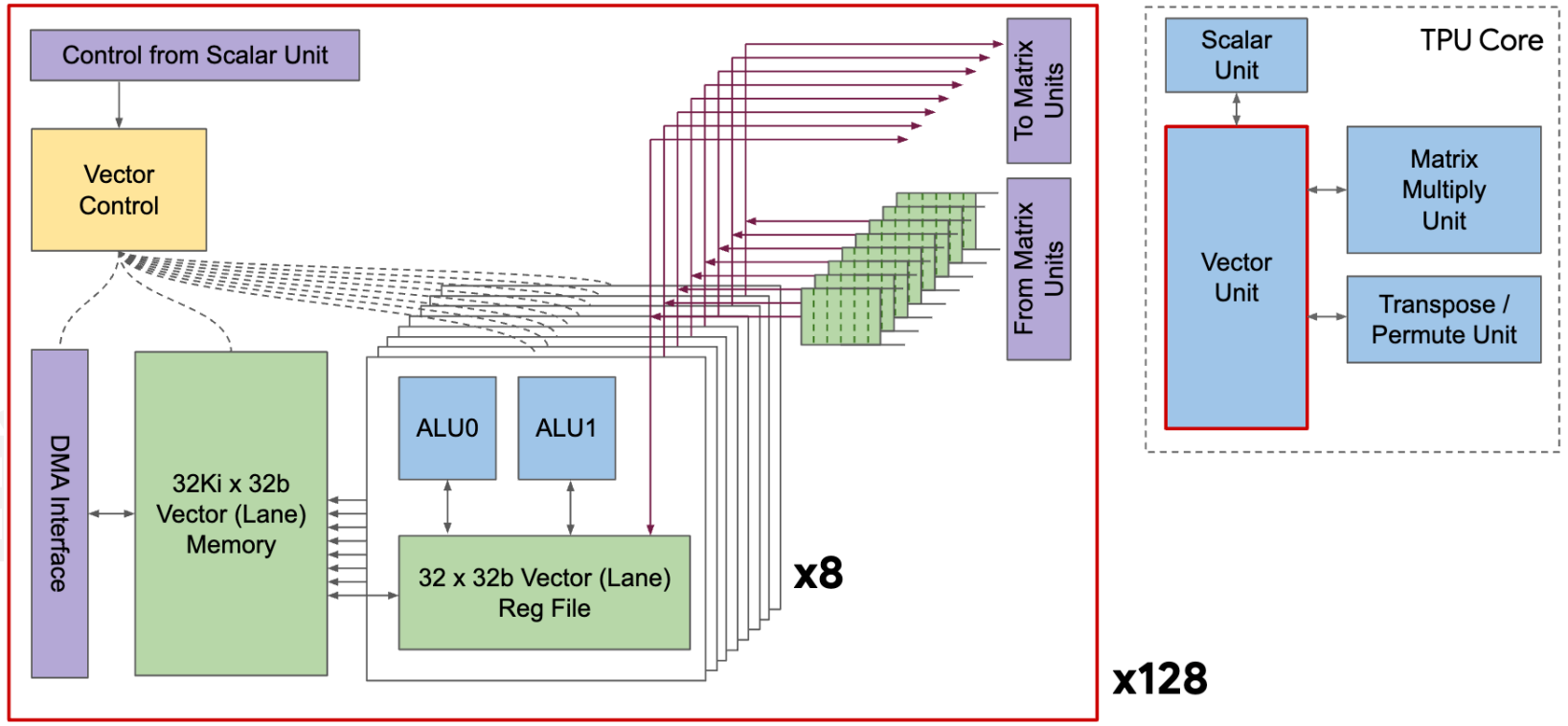
- 采用双核架构，ISA支持矩阵、向量和标量计算，注重通用性
- 矩阵计算单元采用128 x 128的脉动阵列，支持bf16（s1e8m7）乘法和FP32累加
- 为什么要选择128 x 128的脉动阵列，而不是TPU v1中的256 x 256?





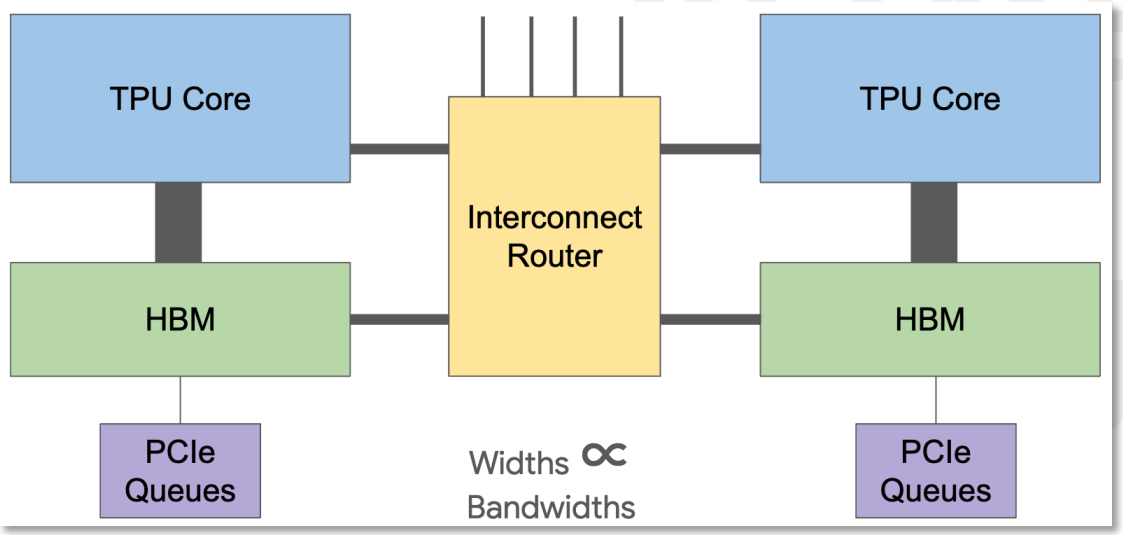
AI加速器架构发展——Google TPU v2

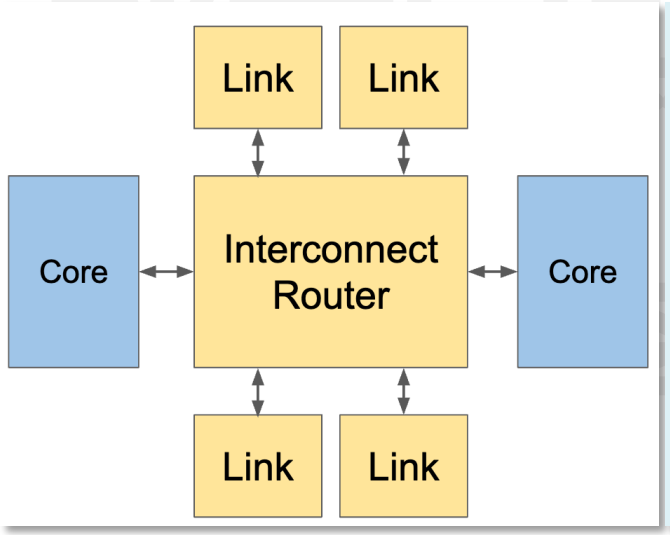
- 向量计算单元的功能：
  - 8个向量计算核，每个核每个周期处理128个输入
  - 为矩阵计算单元提供输入

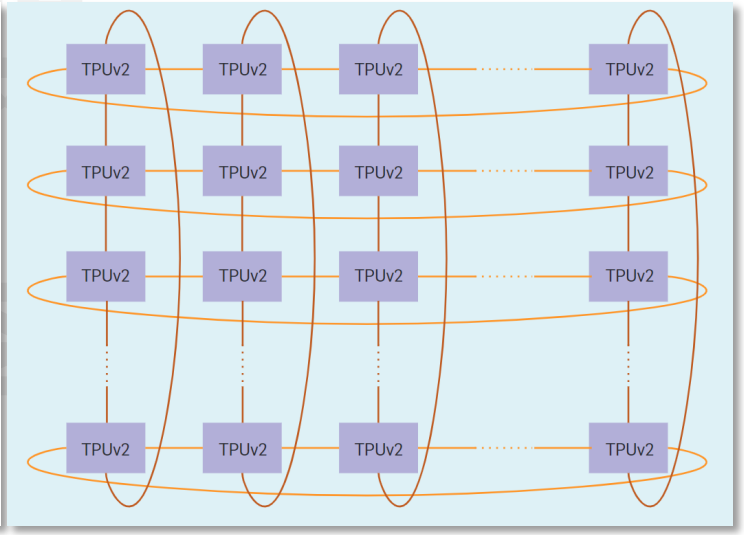


AI加速器架构发展——Google TPU v2

- TPU v2的存储系统
  - 片上存储约为32MB
  - 16GB HBM，带宽约为600 GB/s（TPU v1采用DDR3，带宽约为30 GB/s）
  - 片上路由有4个链路，每个链路带宽为500 Gbps，采用2维圆环（Torus）的拓扑结构
  - 最多支持256个TPU的拓展

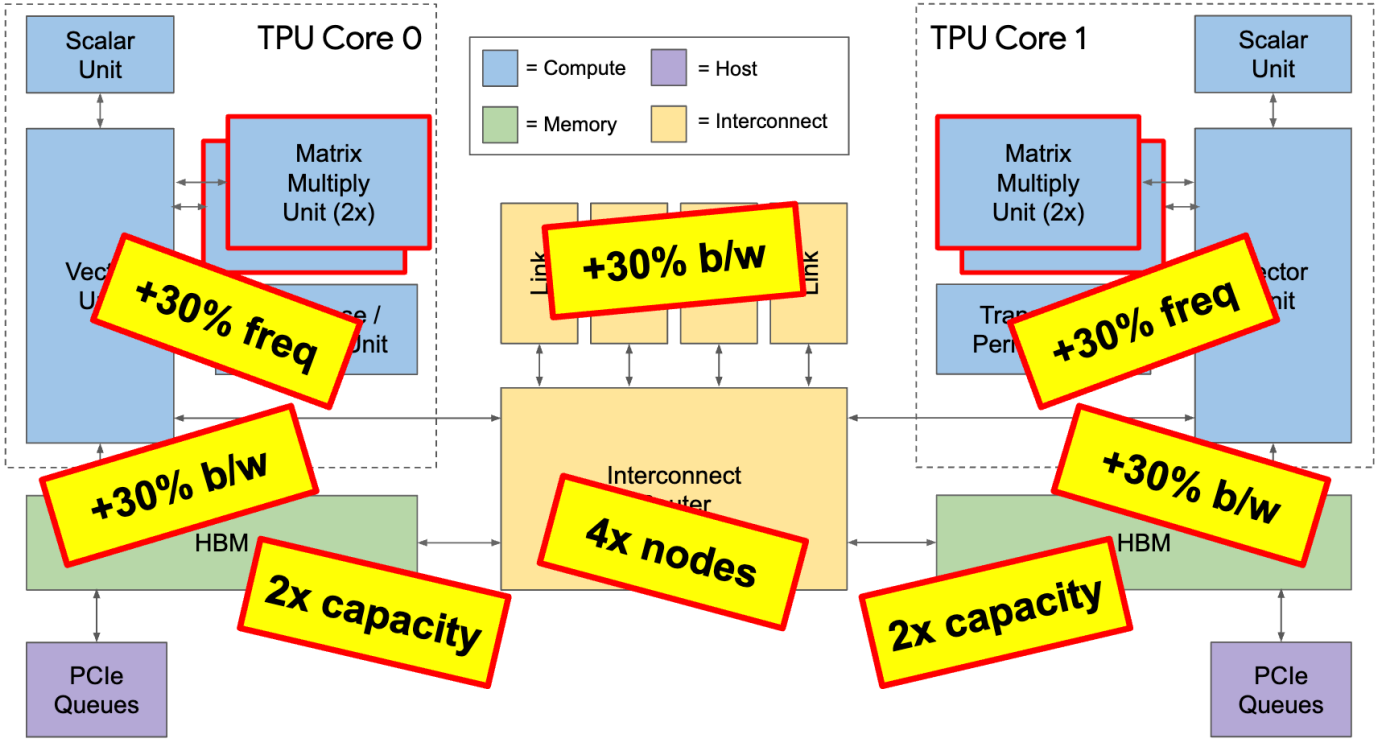






AI加速器架构发展——Google TPU v3

- 在TPU v2的基础上，TPU v3进行了进一步的改进了提升，包括
  - 矩阵乘法单元扩大2倍
  - 通过工程优化，将时钟频率从700 MHz提升到了940 MHz
  - 提升HBM带宽30%，扩大HBM容量2倍，提升互连带宽30%，达到每个链路650 Gb/s
  - 最大支持1024个TPU的扩展



AI加速器架构发展——Google TPU v3

北京大学 PEKING UNIVERSITY

Build it quickly

Achieve high performance...

...at scale

...for new workloads
out-of-the-box

...all while being cost effective

AI加速器架构发展——Google TPU v3

北京大学 PEKING UNIVERSITY

Build it quickly

Achieve high performance...

...at scale

...for new workloads out-of-the-box

...all while being cost effective

- Co-design: simplified hardware with software predictability (e.g., VLIW, scratchpads)
- Willingness to make tradeoffs

AI加速器架构发展——Google TPU v3

北京大学 PEKING UNIVERSITY

Build it quickly

Achieve high performance...

...at scale

...for new workloads out-of-the-box

...all while being cost effective

- Compute density with bfloat16 systolic array
- HBM to feed the compute
- XLA compiler optimizations

AI加速器架构发展——Google TPU v3

北京大学 PEKING UNIVERSITY

Build it quickly

Achieve high performance...

...at scale

...for new workloads out-of-the-box

...all while being cost effective

- System-first approach
- Interconnect with a familiar interface for ease-of-use

AI加速器架构发展——Google TPU v3

北京大学 PEKING UNIVERSITY

Build it quickly

Achieve high performance...

...at scale

...for new workloads out-of-the-box

...all while being cost effective

- Flexible big data cores with principled linear algebra framework
- XLA compiler
- HBM capacity

AI加速器架构发展——Google TPU v3

北京大学 PEKING UNIVERSITY

Build it quickly

Achieve high performance...

...at scale

...for new workloads out-of-the-box

...all while being cost effective

- Matrix Unit efficiency
- Simplicity
- High performance for good perf/$

Lab2简介——选项1：基于CUDA/Triton的GPU算子设计

北京大学 PEKING UNIVERSITY

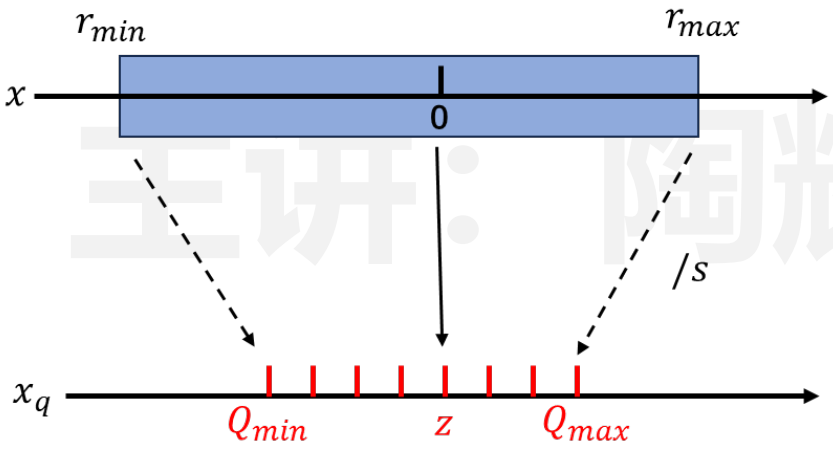
- 在GPU平台为矩阵乘设计量化算子和稀疏算子
- 项目采用Google Colab平台：https://colab.research.google.com/
- 科学上网途径：https://wallesspku.com/blog/
- 实验可在CUDA（C++）和Triton（python）中二选一进行开发
  - CUDA教程：https://developer.nvidia.com/blog/even-easier-introduction-cuda/
  - Triton教程：https://triton-lang.org/main/getting-started/tutorials/index.html
- 如有任何问题，请联系授课老师或助教！

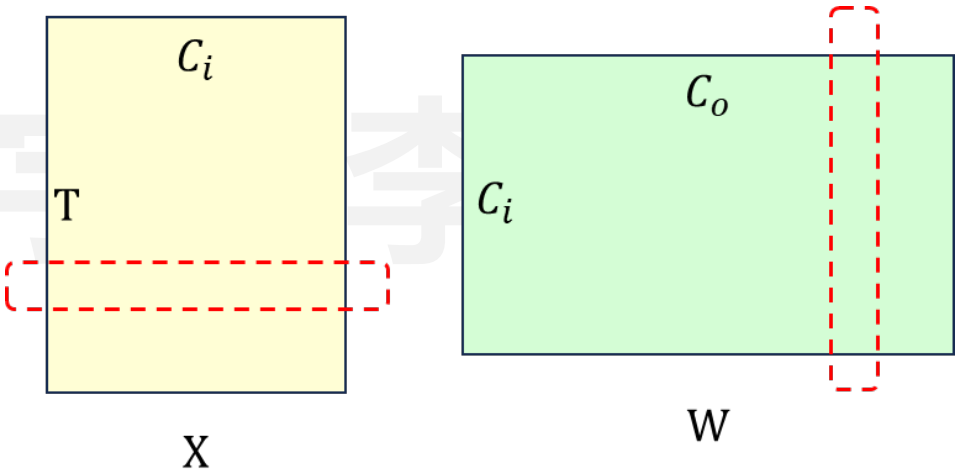
Lab2简介——选项1：基于CUDA/Triton的GPU算子设计

- 量化
  - 量化是指用更少的比特精度（例如INT8，INT4）来表示高精度(FP32/FP16)的权重和激活

$$x_q = clamp\left(round\left(\frac{x}{s}\right); Q_{min}, Q_{max}\right)$$

  - x是高精度的权重或者激活，x_q是量化后的整数。公式1中的s表示缩放因子,负责将$[r_{min}, r_{max}]$映射到$[Q_{min}, Q_{max}]$，公式为$s = \frac{r_{max} - r_{min}}{Q_{max} - Q_{min}} \approx \frac{|r|_{max}}{Q_{max}}$
  - 对于矩阵乘$Y = XW^T$，对X的每一行或者W的每一列计算缩放因子

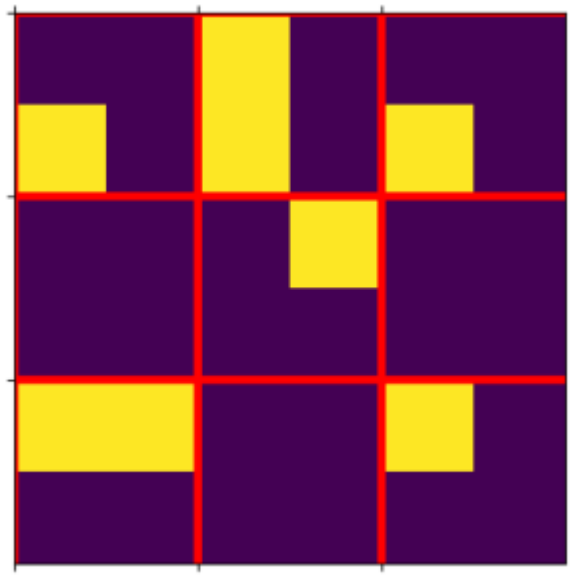




Lab2简介——选项1：基于CUDA/Triton的GPU算子设计

北京大学 PEKING UNIVERSITY

- 稀疏
  - 稀疏矩阵是一种常见的数据结构，其特点在于矩阵中大部分元素为零，而非零元素相对较少
  - 利用矩阵的稀疏性加速矩阵乘
  - 采用pytorch的BSR数据格式：PyTorch 2.1: Quansight's Improvements to BSR Sparse Matrix Multiplication | Quansight Consulting



```
>>> mat = torch.tensor([
    [0, 0, 1, 0, 0, 0],
    [1, 0, 1, 0, 1, 0],
    [0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0],
    [1, 1, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0],
])
>>> mat.to_sparse_csr()
tensor(crow_indices=tensor([0, 1, 4, 5, 5, 8, 8]),
       col_indices=tensor([2, 0, 2, 4, 3, 0, 1, 4]),
       values=tensor([1, 1, 1, 1, 1, 1, 1, 1]),
       size=(6, 6),
       nnz=8,
       layout=torch.sparse_csr)
```

Lab2简介——选项1：基于CUDA/Triton的GPU算子设计

北京大学 PEKING UNIVERSITY

- CUDA
  - 本项目为编写CUDA算子的同学提供了基础矩阵乘编译工具链示例
    - mylinear_cuda_kernel.cu：编写使用cuda进行前向计算的函数
    - mylinear_cuda.cpp：基于 C++ API，调用上述函数
    - setup.py：将C++代码编译为python库
    - cuda.ipynb：调用示例
  - 请同学们参考示例，实现基于CUDA的量化kernel和稀疏kernel
- Triton
  - Triton 是一个由 OpenAI 开发的高效深度学习编程语言和库，旨在简化 GPU 加速的自定义运算核的编写

Lab2简介——选项1：基于CUDA/Triton的GPU算子设计

北京大学 PEKING UNIVERSITY

- 任务说明
  - 本次需要编写W8A8、W4A16的量化算子和fp16的稀疏算子
  - W8A8：完成8bit权重和激活的矩阵乘，它用到的输入是：FP16表示的激活, qweight的转置，input_scale，weight_scale.
  - W4A16：完成4bit权重和16bit激活的矩阵乘，它用到的输入是：FP16表示的激活, qweight, scales.
  - Sparse：完成DDS_MM kernel, 它用到的输入是fp16 dense activation以及 BSR格式表示的weight；这里我们规定weight的维度是[1024, 1024]，需要同学们实现BSR格式中block size为32的DDS_MM kernel

Lab2简介——选项2：面向华为昇腾的算子设计

- 昇腾AI原生算子挑战赛主页：算子挑战赛，校内赛链接
- 基础算子开发命题或算子性能挑战命题都可以
- 鼓励组队，1-3人皆可，如果进入决赛，有额外的分数奖励
- 可以参加北大的校内赛（12月8日比赛），也可以直接参加华为的算子挑战赛