



CS433 Parallel and Distributed Computing

Lecture 10 NPU

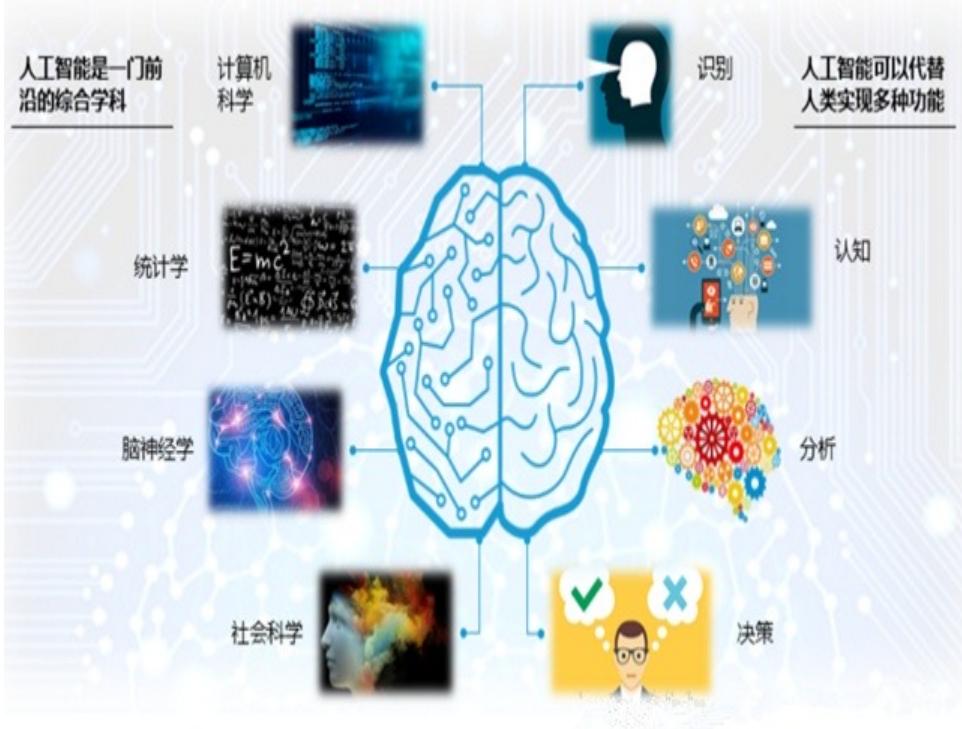
Prof. Xiaoyao Liang

2023/10/27

人工智能基础概览

人工智能 (Artificial Intelligence) :

研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新的技术科学。1956年由约翰·麦卡锡首次提出，当时的定义为“制造智能机器的科学与工程”。人工智能的目的就是让机器能够像人一样思考，让机器拥有智能。时至今日，人工智能的内涵已经大大扩展，



深度强化学习+云计算大数据

- 卷积神经网络模型与参数训练技巧的进步；
- 硬件的进步：摩尔定律，可观的计算能力，云计算；
- 互联网+海量大数据集，深度学习的准确度是随着数据的增长而增加。

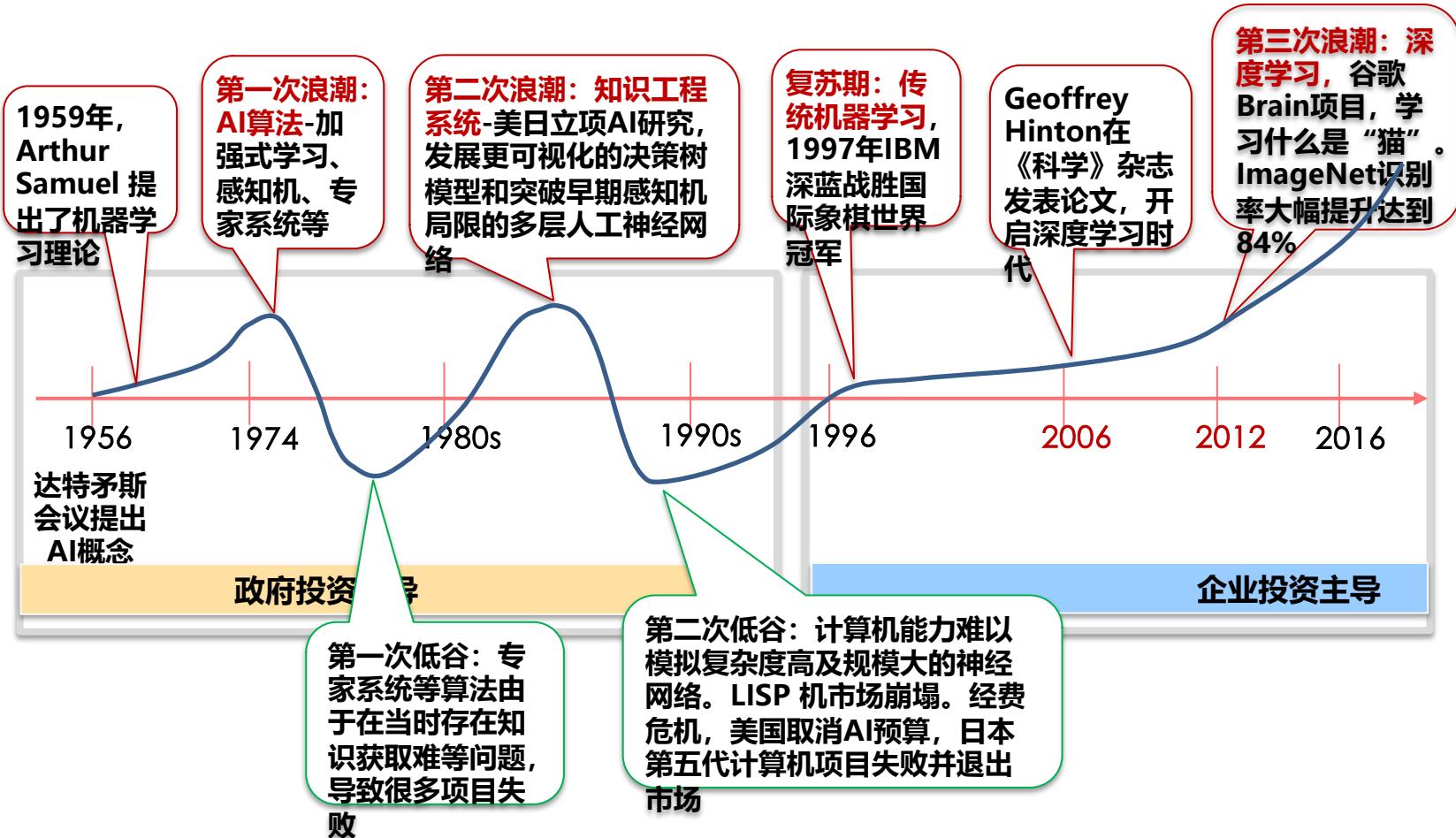
知识工程+机器学习

- 知识工程体现为各种**专家系统**，比如医学、工程学，知识的总结与获取较难，部分专家不愿意分享经验；
- 将各种**机器学习算法**引入人工智能，让机器**从数据中自动学习**，获得知识。

符号主义+逻辑方法

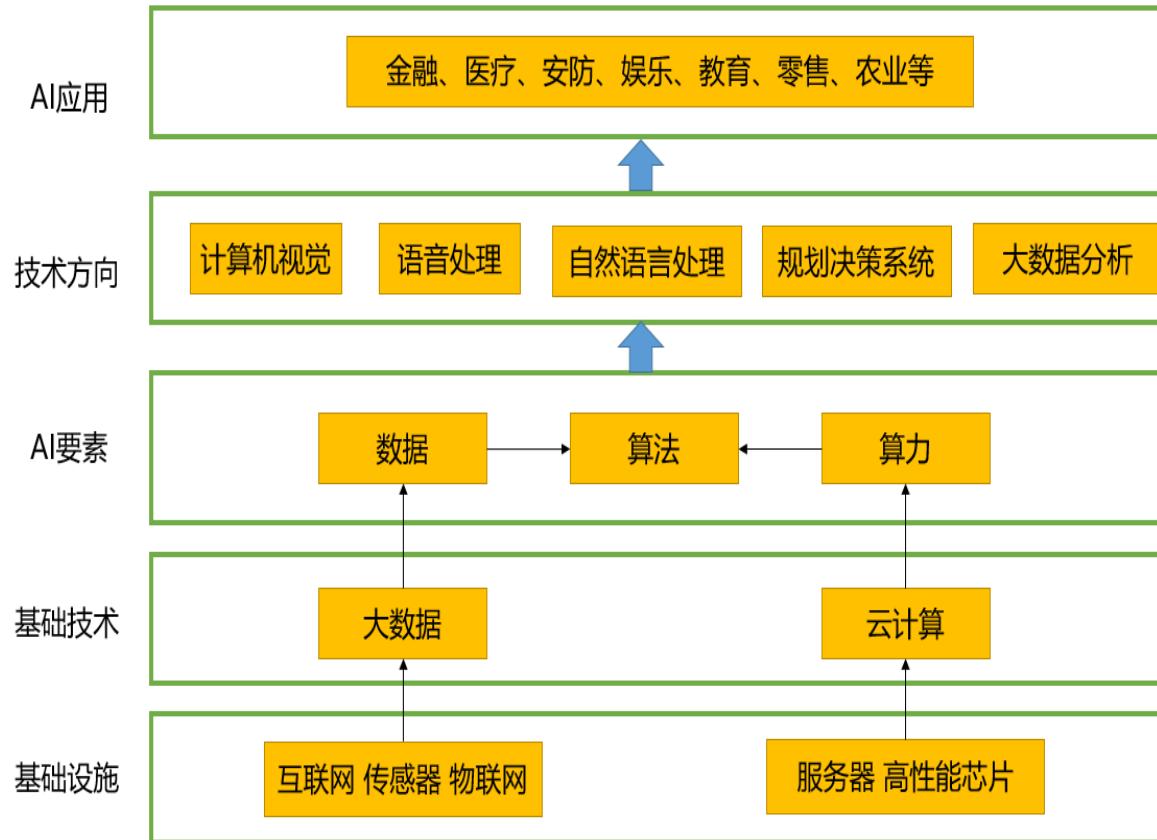
- 基于**决策论的逻辑主义推理方法**；
- 用机器证明的办法去推理知识，如通过**逻辑程序语言**证明数学定理；
- 统计方法中，**引入符号主义进行语义处理**，实现人机交互。

人工智能发展历史



人工智能产业生态

人工智能的四要素是数据、算法、算力、场景。要满足这四要素，我们需要将AI与云计算、大数据和物联网结合以服务于智能社会。



人工智能应用技术方向

- 现在AI的应用技术方向主要分为：

图像识别（>96.5%）：

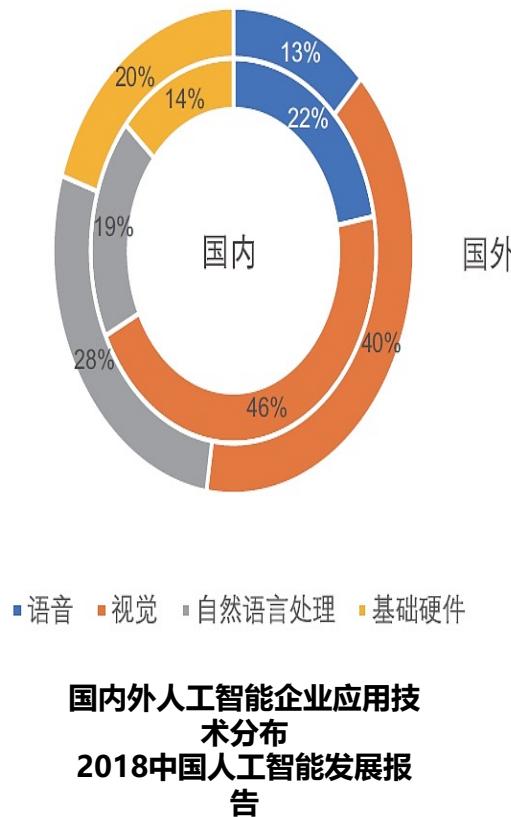
- 人脸识别（99.8%，人眼97.5%）
- 车牌（99.7%）、商标识别等
- 图片搜索（>80%）、分类等
- 唇语（92%，远超人类专家）

语音，语言，语义识别：

- 语音识别（97% ~ 99%）
- 语音-文本转换（>95%）
- 自动翻译（>90% 英文<->德、法等）
- 个人助理/ChatBot
- 语调情感识别

其他应用：

- 智能库存管理
- 疾病辅助诊疗
- DC自动运维
- 商品智能推荐
-



人工智能当前所处的阶段 —— 感知智能的初级阶段

人工
智
能
三
阶
段



计算智能

表现

能存会算：机器开始像人类一样会计算，传递信息

示例

例：分布式计算、神经网络

价值

价值：能够帮助人类存储和快速处理海量数据，是感知和认知的基础



感知智能

能听会看：机器开始看懂和听懂，做出判断，采取一些简单行动

例：可以识别人脸的摄像头、可以听懂语言的音箱

价值：能够帮助人类高效地完成“看”和“听”相关的工作



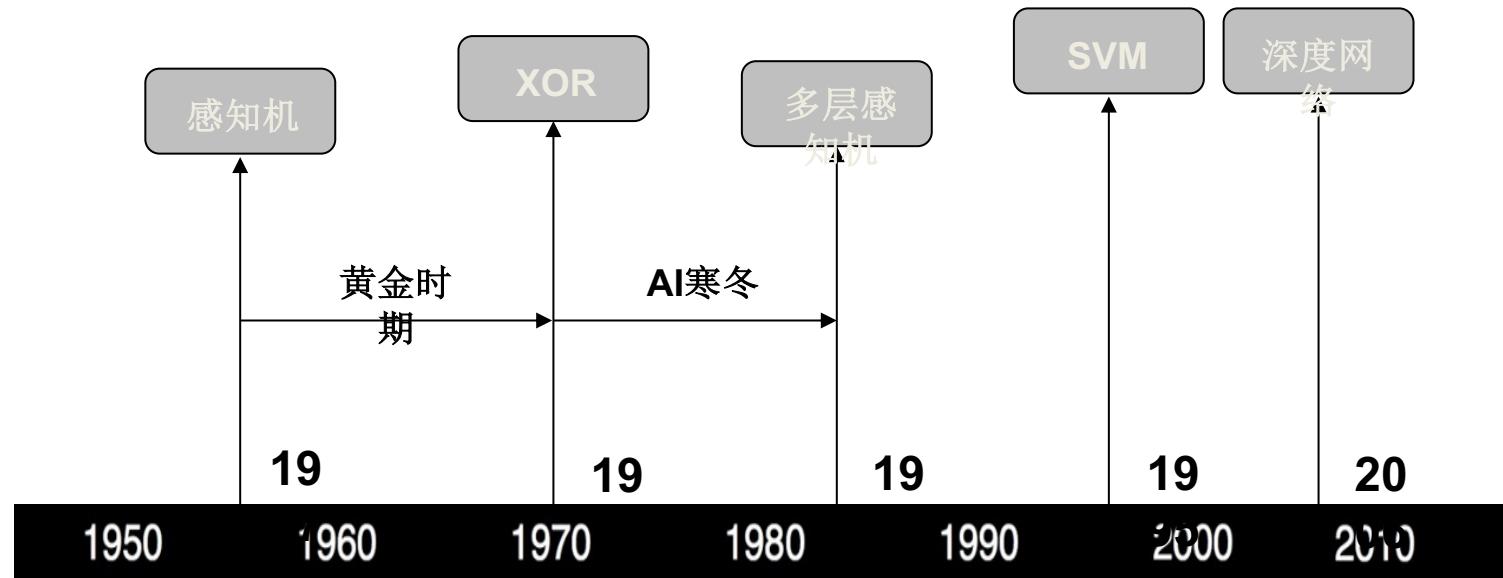
认知智能

能理解会思考：机器开始像人类一样能理解、思考与决策

例：完全独立驾驶的无人驾驶汽车、自主行动的机器人

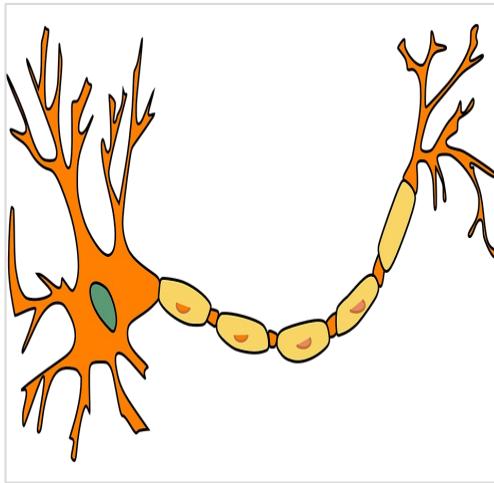
AI的发展目前处于感知智能的初级阶段

深度学习发展

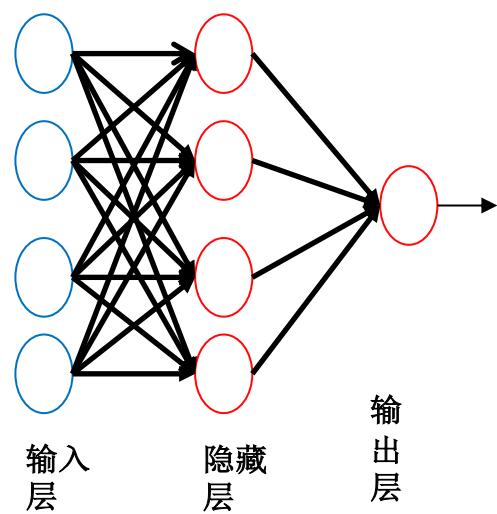


神经网络基础

- 人工神经网络，简称神经网络（Artificial Neural Network, ANN）：是由人工神经元互连组成的网络，它是从微观结构和功能上对人脑的抽象、简化，是模拟人类智能的一条重要途径，反映了人脑功能的若干基本特征，如并行信息处理、学习、联想、模式分类、记忆等。
- 隐藏层比较多（大于2）的神经网络叫做深度神经网络（Deep Neural Network, DNN）。而深度学习，就是使用深度神经网络架构的机器学习方法。



人类神经网络



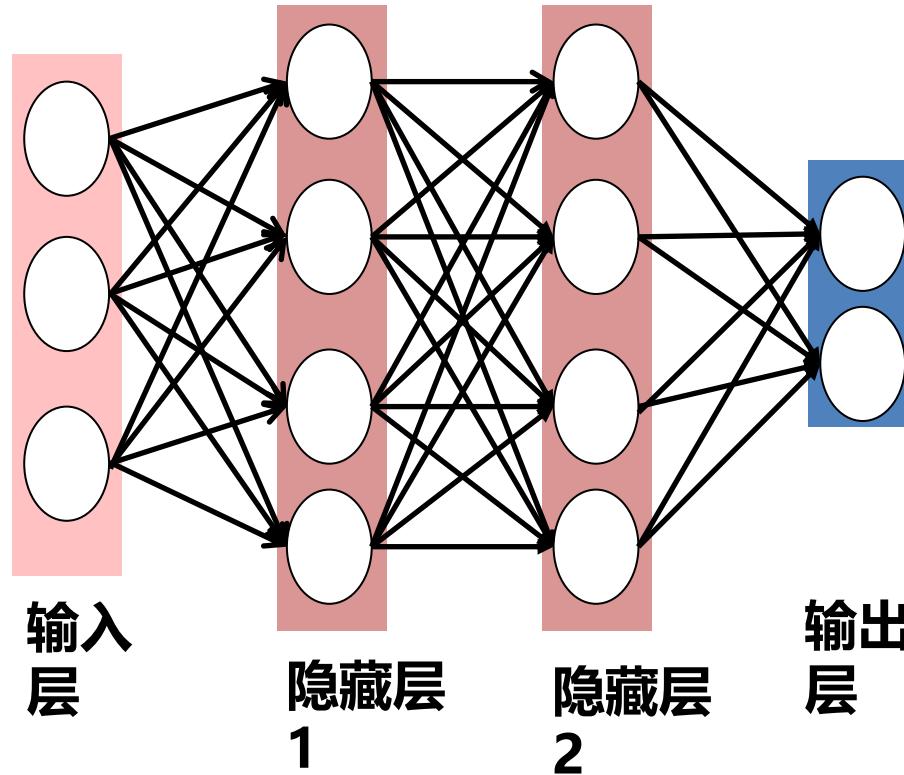
感知器



深度神经网络

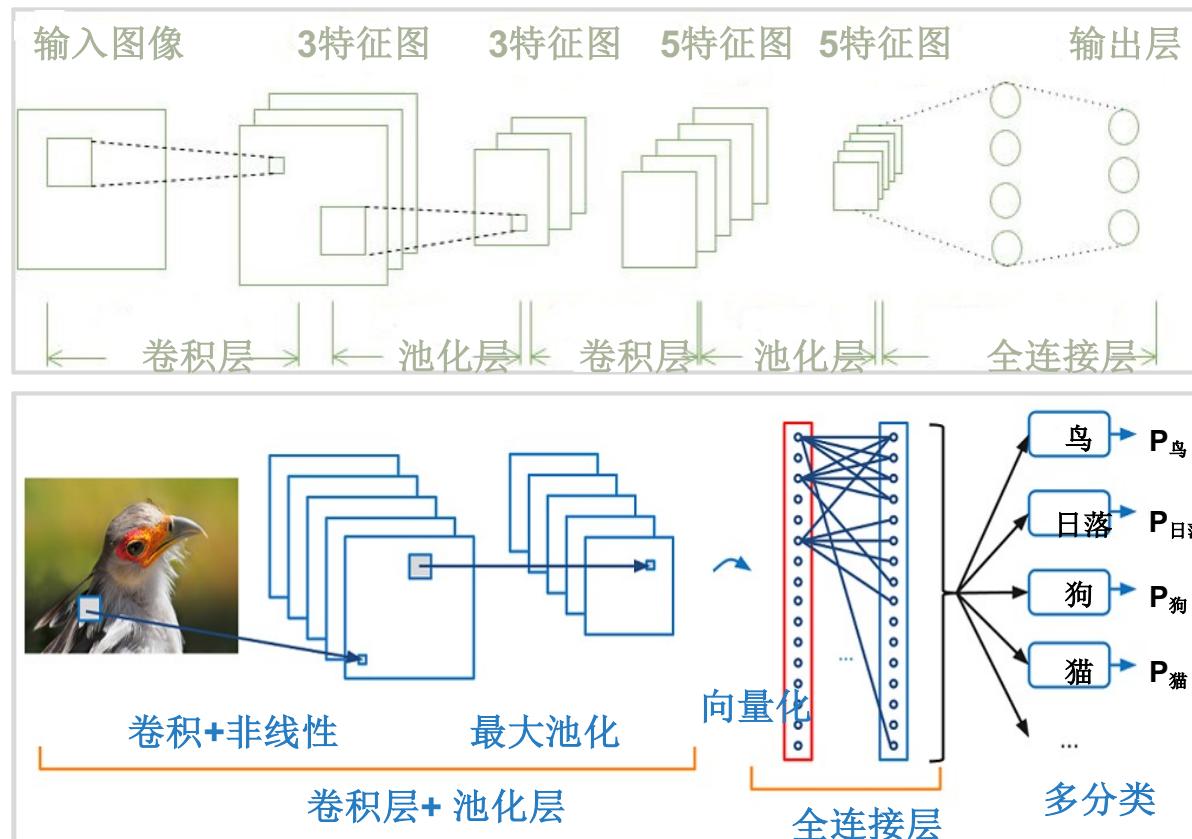
前馈神经网络

- 前馈神经网络是一种最简单的神经网络，各神经元分层排列。每个神经元只与前一层的神经元相连，同一层的神经元之间没有互相连接，层间信息的传送只沿一个方向进行。



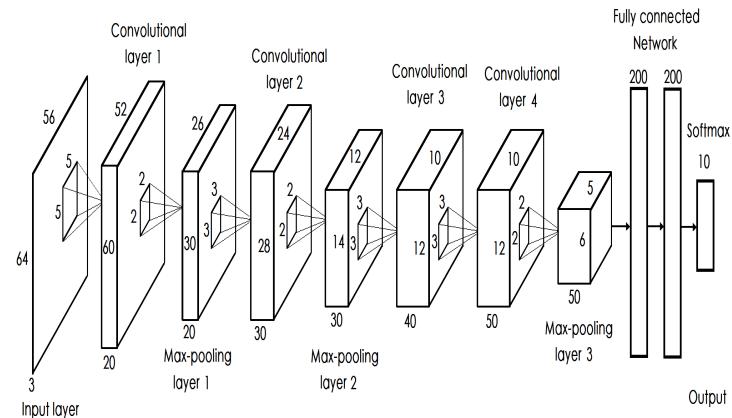
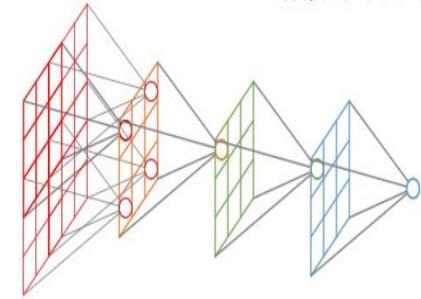
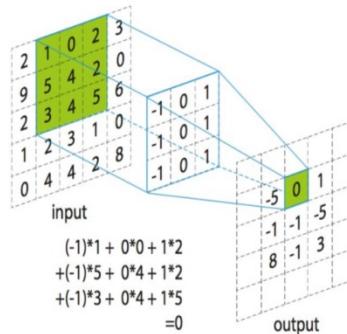
卷积神经网络

- 卷积神经网络 (Convolutional Neural Network, CNN) 是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于图像处理有出色表现。它包括卷积层(convolutional layer)，池化层(pooling layer)和全连接层(fully_connected layer)。



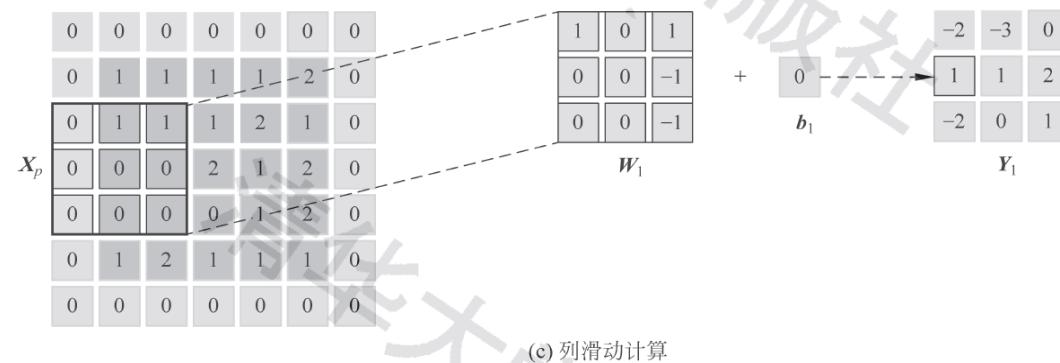
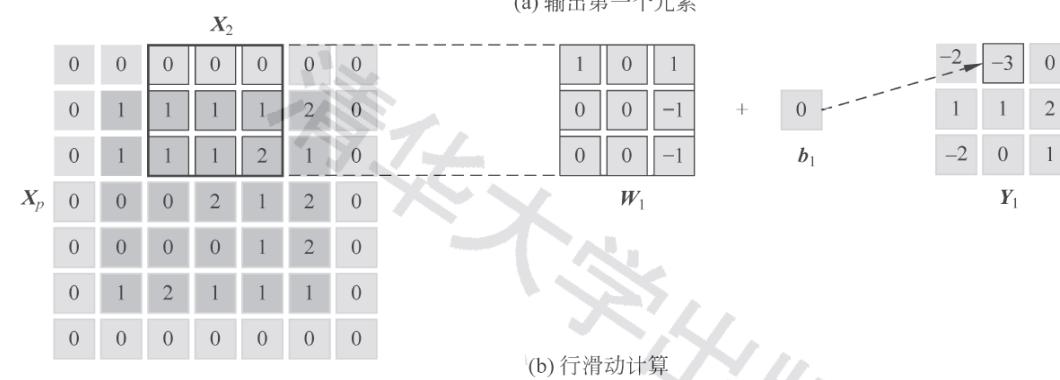
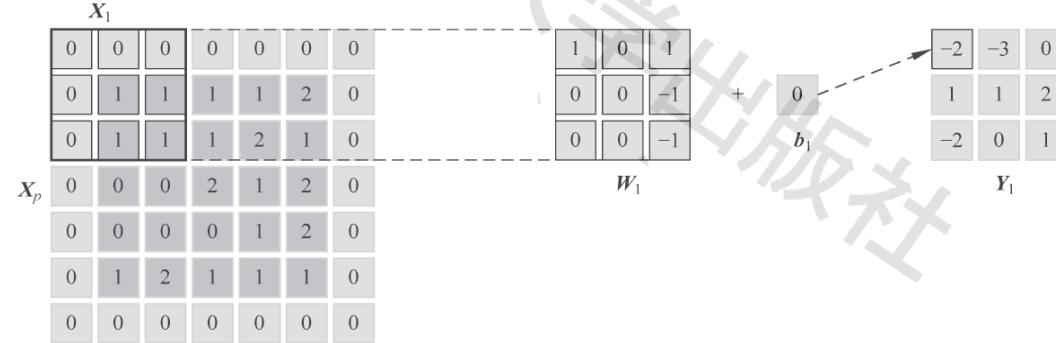
卷积神经网络的重要概念

- **卷积核 (Convolution Kernel)**，根据一定规则进行图片扫描并进行卷积计算的对象称为卷积核。卷积核可以提取局部特征。
- **卷积核尺寸 (Kernel Size)**，卷积核是一个3维的矩阵，可以用一个立方体图示，宽w，高h，深度d。深度d由输入的通道数决定，一般描述卷积核尺寸时，可以只描述宽w和高h。
- **特征图 (Feature Map)**，经过卷积核卷积过后得到的结果矩阵就是特征图。每一个卷积核会得到一层特征图，有多个卷积核则会得到多层的卷积图。
- **特征图尺寸 (Feature Map Size)**，特征图也是一个3维的矩阵，可以用一个立方体图示，宽w，高h，深度d。深度d由当前层的卷积核个数决定，一般描述特征图尺寸时，可以只描述宽w和高h
- **步长 (stride)**，卷积核在输入图像上滑动的跨度。如果卷积核一次移动一个像素，我们称其步长为1。
- **零填充 (padding)**，为了提取图像边缘的信息，并且保证输出特征图的尺寸满足要求，可以对输入图像边缘填充一个全为0的边框，边框的像素宽度就是padding。
- local receptive field，权值共享，多卷积核（滤波器）
- **结果：**通过卷积获得了特征 (features) 。



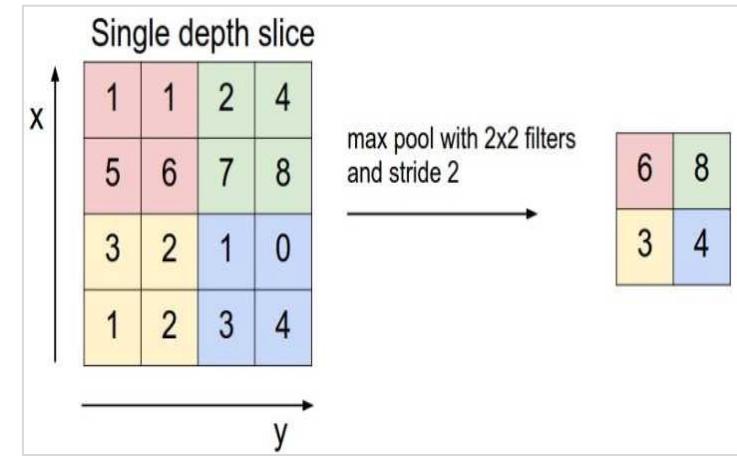
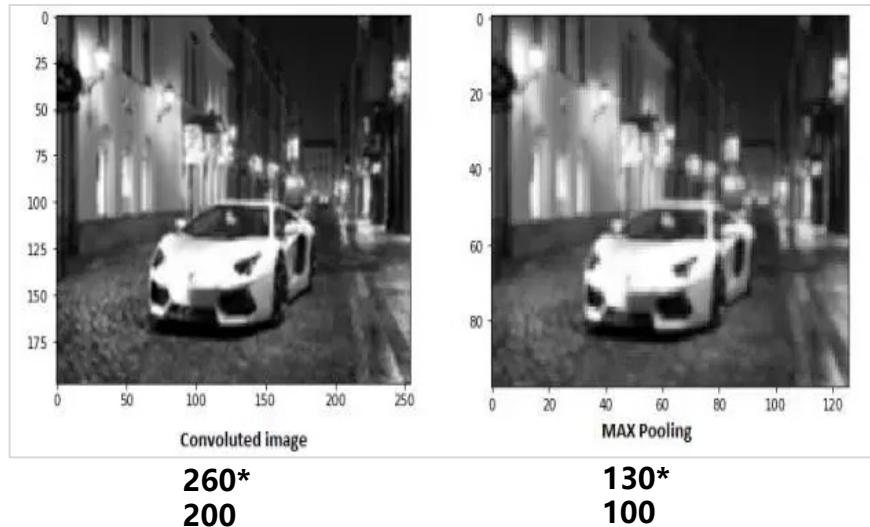


卷积



池化

- 有时图像太大，我们需要减少训练参数的数量，它被要求在随后的卷积层之间周期性地引进池化层。池化层一般分为最大池化 (max pooling) 和平均池化 (mean pooling) 。
- 目的：不仅具有低得多的维度 (相比使用所有提取得到的特征)，同时还会改善结果(不容易过拟合)

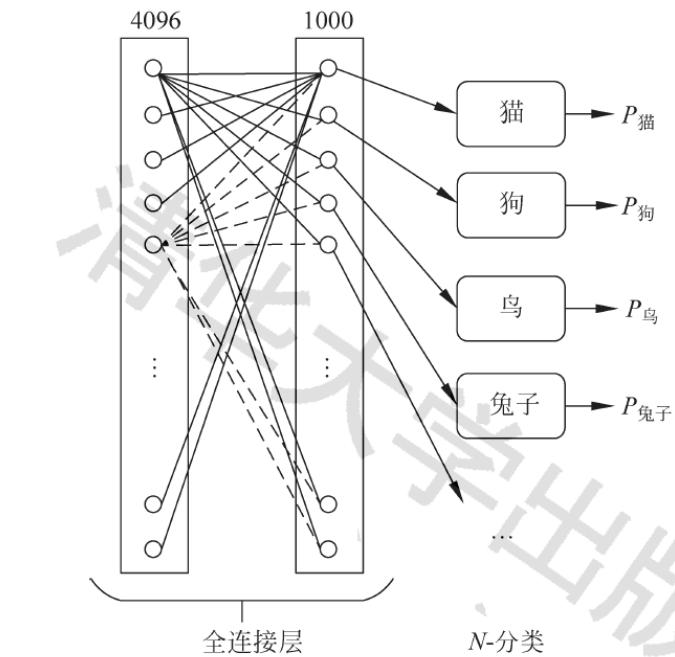
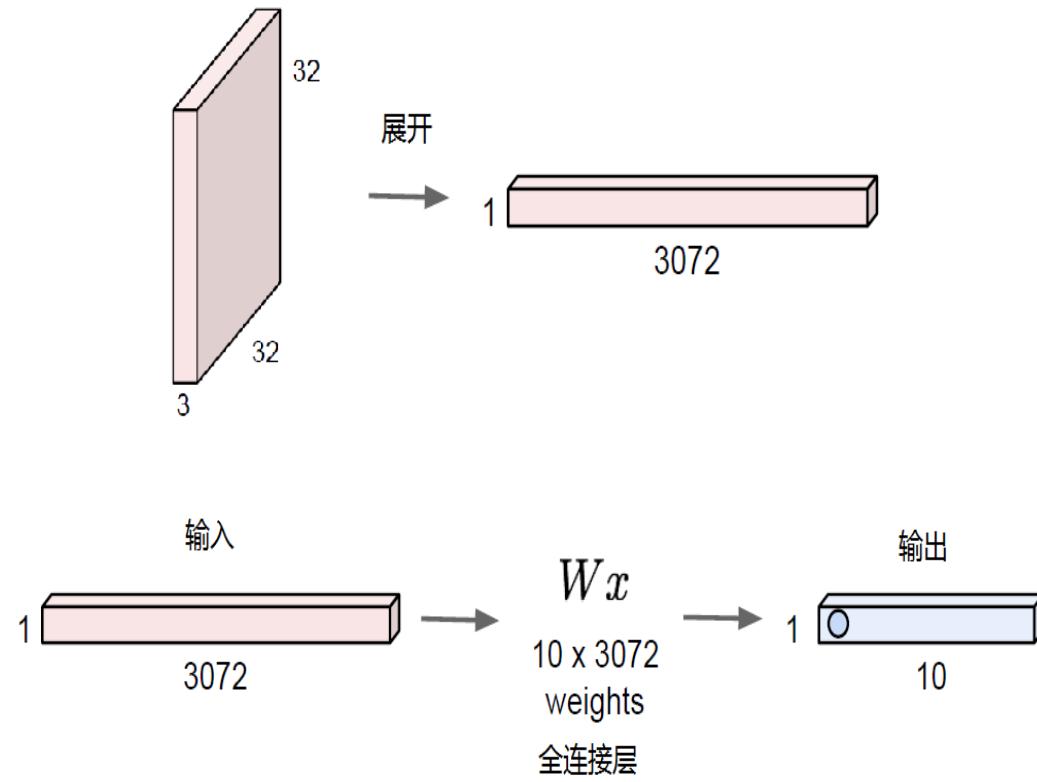


池化的
实现



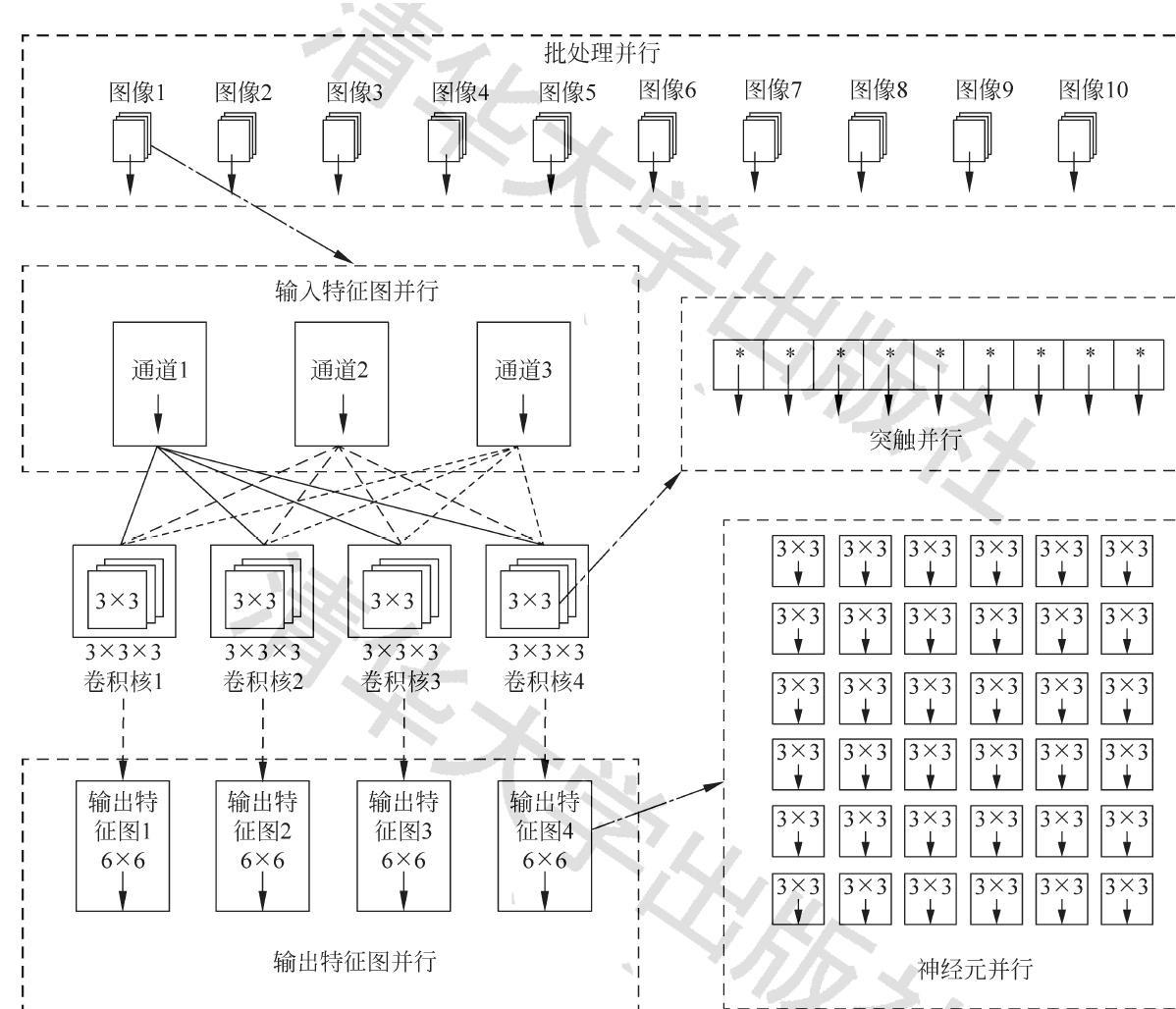
全连接

- 全连接层可以用来将最后得到的特征映射到线性可分的空间，通常，卷积神经网络会将末端得到的特征图平摊成一个长的列向量，经过全连接层的计算得到最终的输出层。





并行度



$$\text{最大并行度} = K \times K \times S_o \times S_o \times N \times M \times P$$



并行度

K:卷积核维度=3

N:通道数 =3

M:卷积核个数 =4

S0:输出特征图维度=6

P:批处理图像数=10

$$3 \times 3 \times 6 \times 6 \times 3 \times 4 \times 10 = 38880$$



AI芯片分类

从技术架构来看，大致分为四个类型：

- CPU (Central Processing Unit, 中央处理器)：是一块超大规模的集成电路，是一台计算机的运算核心 (Core) 和控制核心 (Control Unit)。它的功能主要是解释计算机指令以及处理计算机软件中的数据。
- GPU (Graphics Processing Unit, 图形处理器)：又称显示核心、视觉处理器、显示芯片，是一种专门在个人电脑、工作站、游戏机和一些移动设备（如平板电脑、智能手机等）上图像运算工作的微处理器。
- ASIC (Application Specific Integrated Circuit, 专用集成电路)：适合于某一单一用途的集成电路产品。
- FPGA (Field Programmable Gate Array, 现场可编程门阵列)：其设计初衷是为了实现半定制芯片的功能，即硬件结构可根据需要实时配置灵活改变。

从功能来看，可以分为Training（训练）和Inference（推理）两个类型：

- Training环节通常需要通过大量的数据输入，或采取增强学习等非监督学习方法，训练出一个复杂的深度神经网络模型。训练过程涉及海量的训练数据和复杂的深度神经网络结构，运算量巨大，需要庞大的计算规模，对于处理器的计算能力、精度、可扩展性等性能要求很高。常用的例如NVIDIA的GPU集群、Google的TPU等。
- Inference环节指利用训练好的模型，使用新的数据去“推理”出各种结论，如视频监控设备通过后台的深度神经网络模型，判断一张抓拍到的人脸是否属于特定的目标。虽然Inference的计算量相比Training少很多，但仍然涉及大量的矩阵运算。在推理环节，GPU、FPGA和ASIC都有很多应用价值。



AI芯片生态

CPU

- 增加指令（修改架构）的方式提升AI性能
- Intel (CISC架构) 加入AVX512等指令，在ALU计算模块加入矢量运算模块 (FMA)
- ARM (RISC架构) 加入Cortex A等指令集并计划持续升级
- 增加核数提升性能，但带来功耗和成本增加
- 提高频率提升性能，但提升空间有限，同时高主频会导致芯片出现功耗过大和过热问题

GPU

- 异构计算的主力，AI计算兴起之源，生态成熟
- Nvidia沿用GPU架构，对深度学习主要向两个方向发力：
 - 丰富生态：推出cuDNN针对神经网络的优化库，提升易用性并优化GPU底层架构
 - 提升定制性：增加多数据类型支持（不再坚持float32，增加int8等）；添加深度学习专用模块（如V100的TensorCore）
- 当前主要问题在于：成本高，能耗比低，延迟高

FPGA

- 采用HDL可编程方式，灵活性高，可重构（烧），可深度定制
- 可通过多片FPGA联合将DNN模型加载到片上进行低延迟计算，计算性能优于GPU，但由于需考虑不断擦写，性能达不到最优（冗余晶体管和连线，相同功能逻辑电路占芯片面积更大）
- 由于可重构，供货风险和研发风险较低，成本取决于购买数量，相对自由
- 设计、流片过程解耦，开发周期较长（通常半年），门槛高

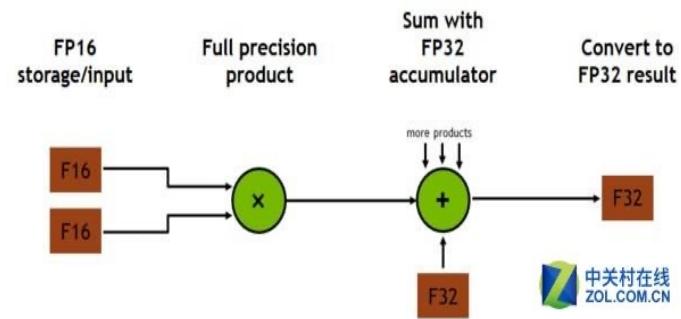
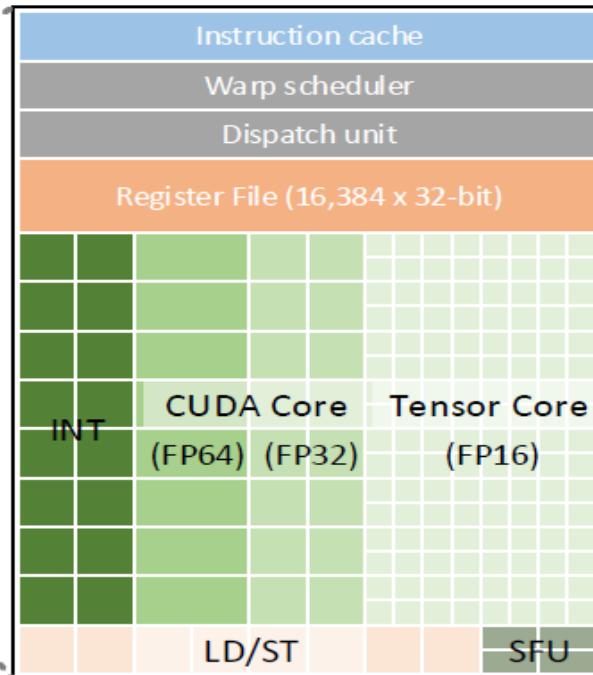
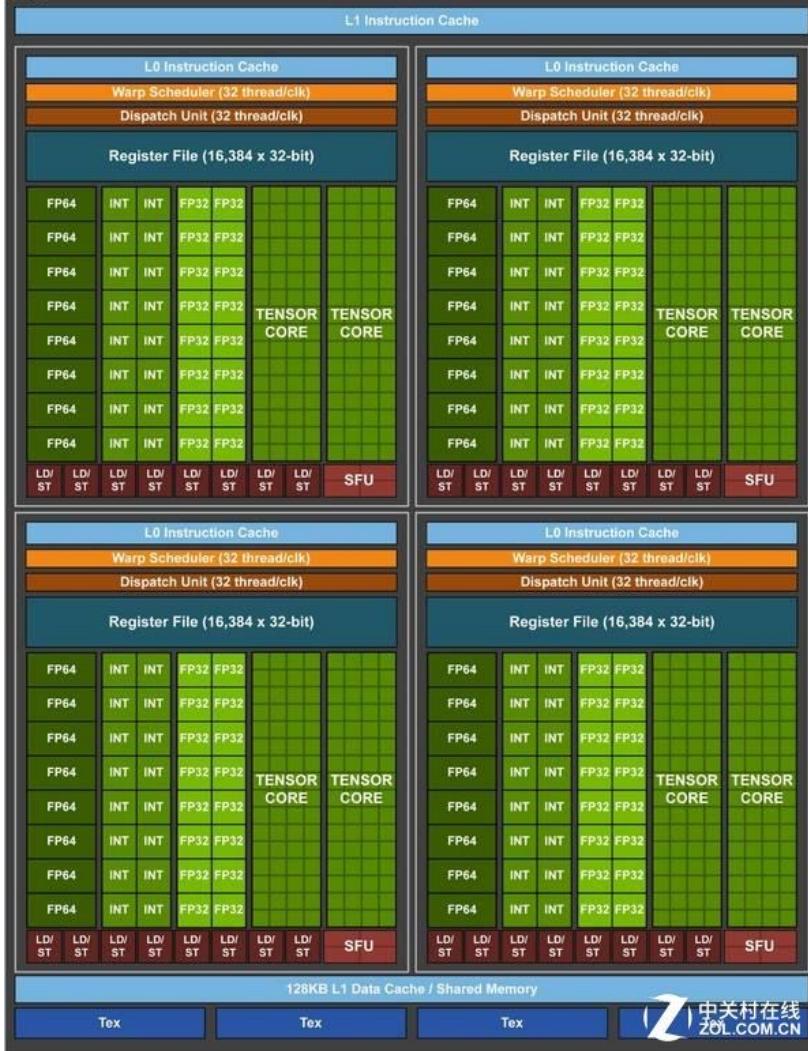
ASIC

- 不考虑带宽、功耗约束，可按需定制运算单元数目和物理空间分布，计算性能和能耗效率极高，业务范围窄（需求明确）
- 成本受应用规模约束，制造成本高（单片成本低，但厂商只接收一定数量规模定制），发生错误只能重头开始，风险高
- 开发周期长，18-24月（TPU团队15月），立项到上线时间长
- 完整的产业链下，价值不在于本身，而是领域广张走向上下游必经之路



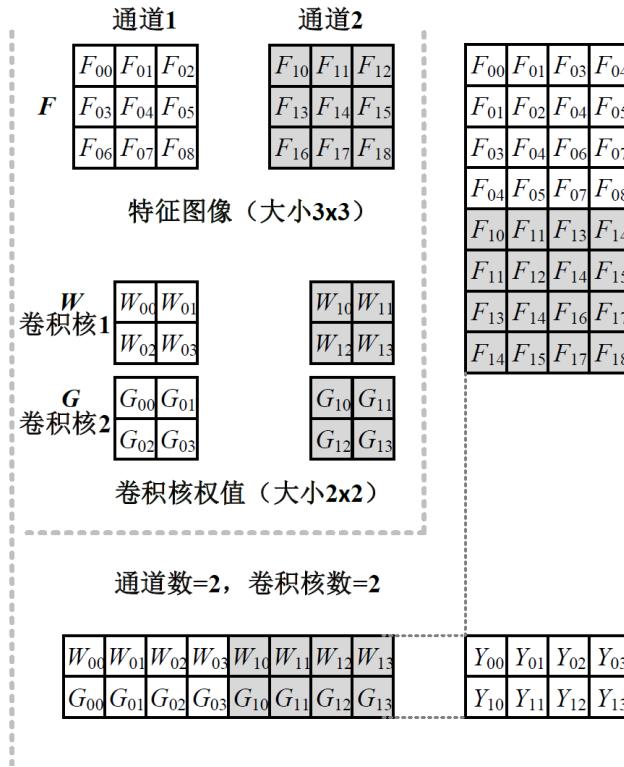
NVIDIA Volta

SM

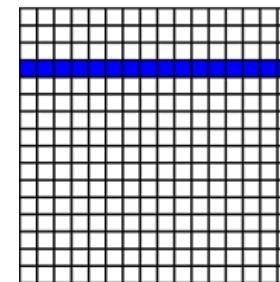




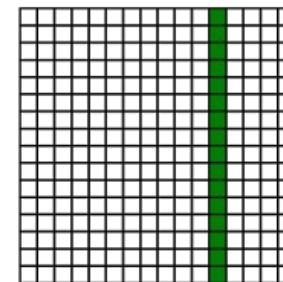
GPU做卷积



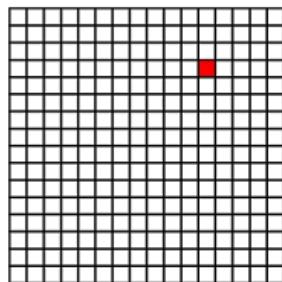
A



B



C



GEMM

Img2Col

Hardware	Applied Algorithm	Called Function	Execution Time	Ratio
CUDA Core	GEMM	scudnn_128x64	3722.66	98%
	Im2col	computeOffsets	75.97	2%
	Winograd Index	winograd_128x128	2941.87	85%
		generateWinograd	519.15	15%
Tensor Core	GEMM	s884cudnn_fp16	619.22	86%
	Data Transform	nchwToNhwc/nhwcToNchw	79.27	11%
	Im2col	computeOffsets	21.61	3%
	Winograd Index	winograd_128x128	546.44	70%
		generateWinograd	234.19	30%



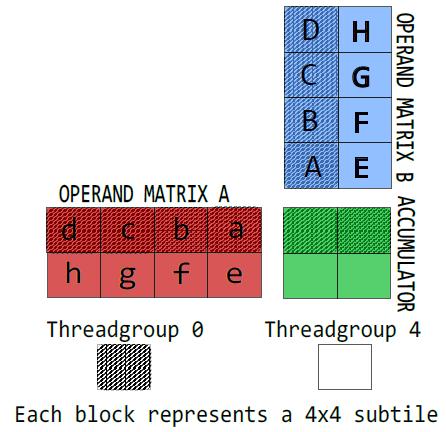
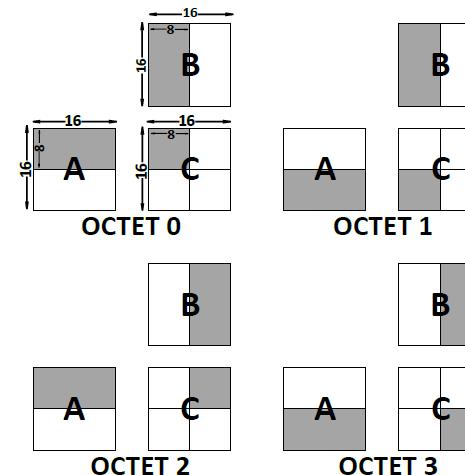
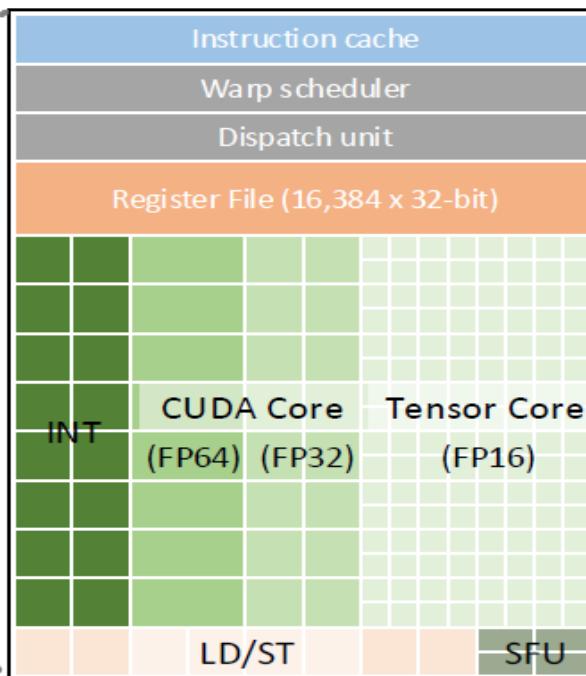
GPU做卷积

周期	线程1	线程2	…	线程8
1	$PS_1 = W_{00} \cdot F_{00} + 0$	$PS_2 = W_{00} \cdot F_{01} + 0$	…	$PS_8 = G_{00} \cdot F_{04} + 0$
2	$PS_1 = W_{01} \cdot F_{01} + PS_1$	$PS_2 = W_{01} \cdot F_{02} + PS_2$	…	$PS_8 = G_{01} \cdot F_{05} + PS_8$
3	$PS_1 = W_{02} \cdot F_{03} + PS_1$	$PS_2 = W_{02} \cdot F_{04} + PS_2$	…	$PS_8 = G_{02} \cdot F_{07} + PS_8$
4	$PS_1 = W_{03} \cdot F_{04} + PS_1$	$PS_2 = W_{03} \cdot F_{05} + PS_2$	…	$PS_8 = G_{03} \cdot F_{08} + PS_8$
5	$PS_1 = W_{10} \cdot F_{10} + PS_1$	$PS_2 = W_{10} \cdot F_{11} + PS_2$	…	$PS_8 = G_{10} \cdot F_{14} + PS_8$
6	$PS_1 = W_{11} \cdot F_{11} + PS_1$	$PS_2 = W_{11} \cdot F_{12} + PS_2$	…	$PS_8 = G_{11} \cdot F_{15} + PS_8$
7	$PS_1 = W_{12} \cdot F_{13} + PS_1$	$PS_2 = W_{12} \cdot F_{14} + PS_2$	…	$PS_8 = G_{12} \cdot F_{17} + PS_8$
8	$PS_1 = W_{13} \cdot F_{14} + PS_1$	$PS_2 = W_{13} \cdot F_{15} + PS_2$	…	$PS_8 = G_{13} \cdot F_{18} + PS_8$



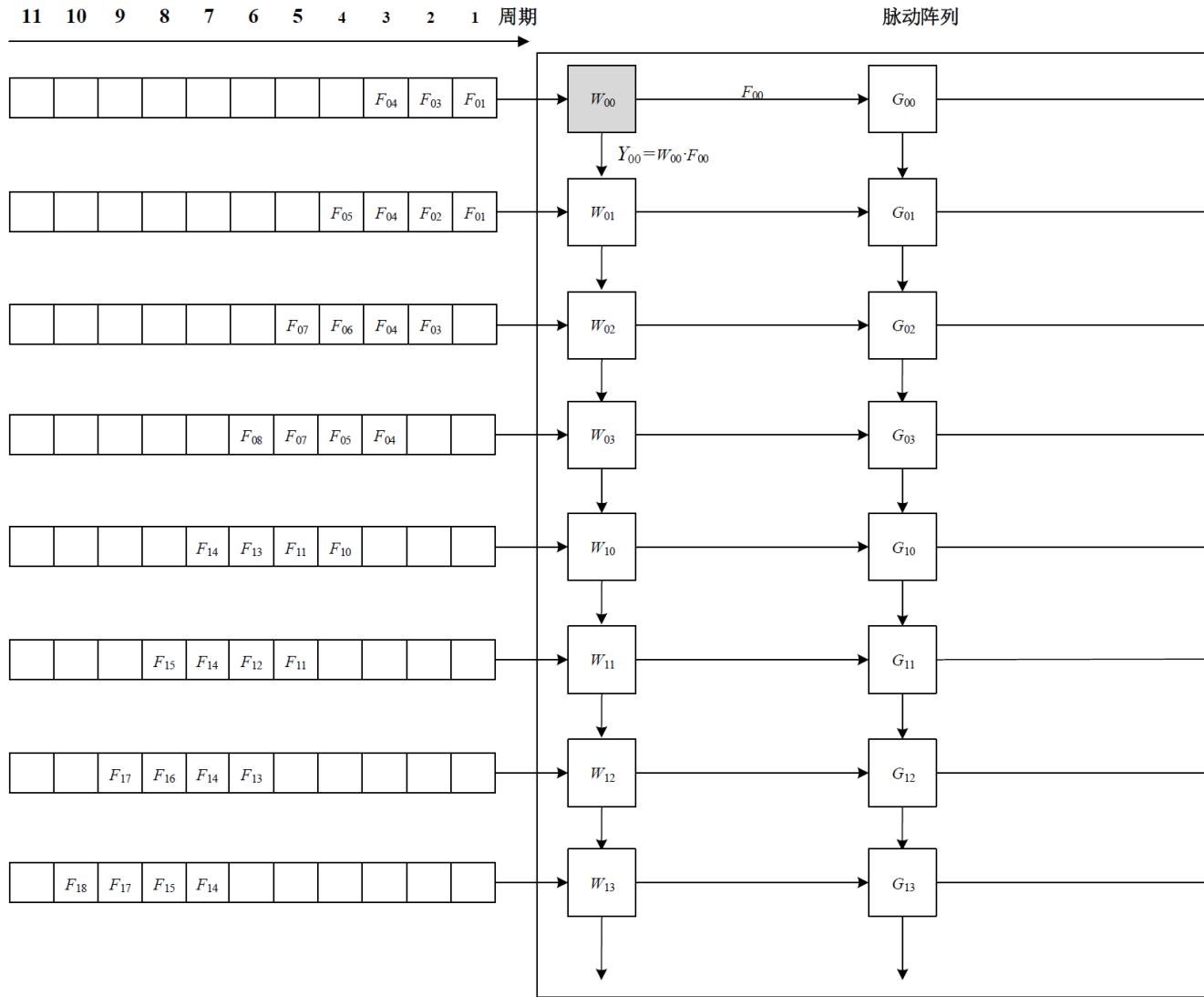
Tensor Core

$$\begin{pmatrix} D_{00} & D_{01} & D_{02} & D_{03} \\ D_{10} & D_{11} & D_{12} & D_{13} \\ D_{20} & D_{21} & D_{22} & D_{23} \\ D_{30} & D_{31} & D_{32} & D_{33} \end{pmatrix} = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix} \times \begin{pmatrix} B_{00} & B_{01} & B_{02} & B_{03} \\ B_{10} & B_{11} & B_{12} & B_{13} \\ B_{20} & B_{21} & B_{22} & B_{23} \\ B_{30} & B_{31} & B_{32} & B_{33} \end{pmatrix} + \begin{pmatrix} C_{00} & C_{01} & C_{02} & C_{03} \\ C_{10} & C_{11} & C_{12} & C_{13} \\ C_{20} & C_{21} & C_{22} & C_{23} \\ C_{30} & C_{31} & C_{32} & C_{33} \end{pmatrix}$$



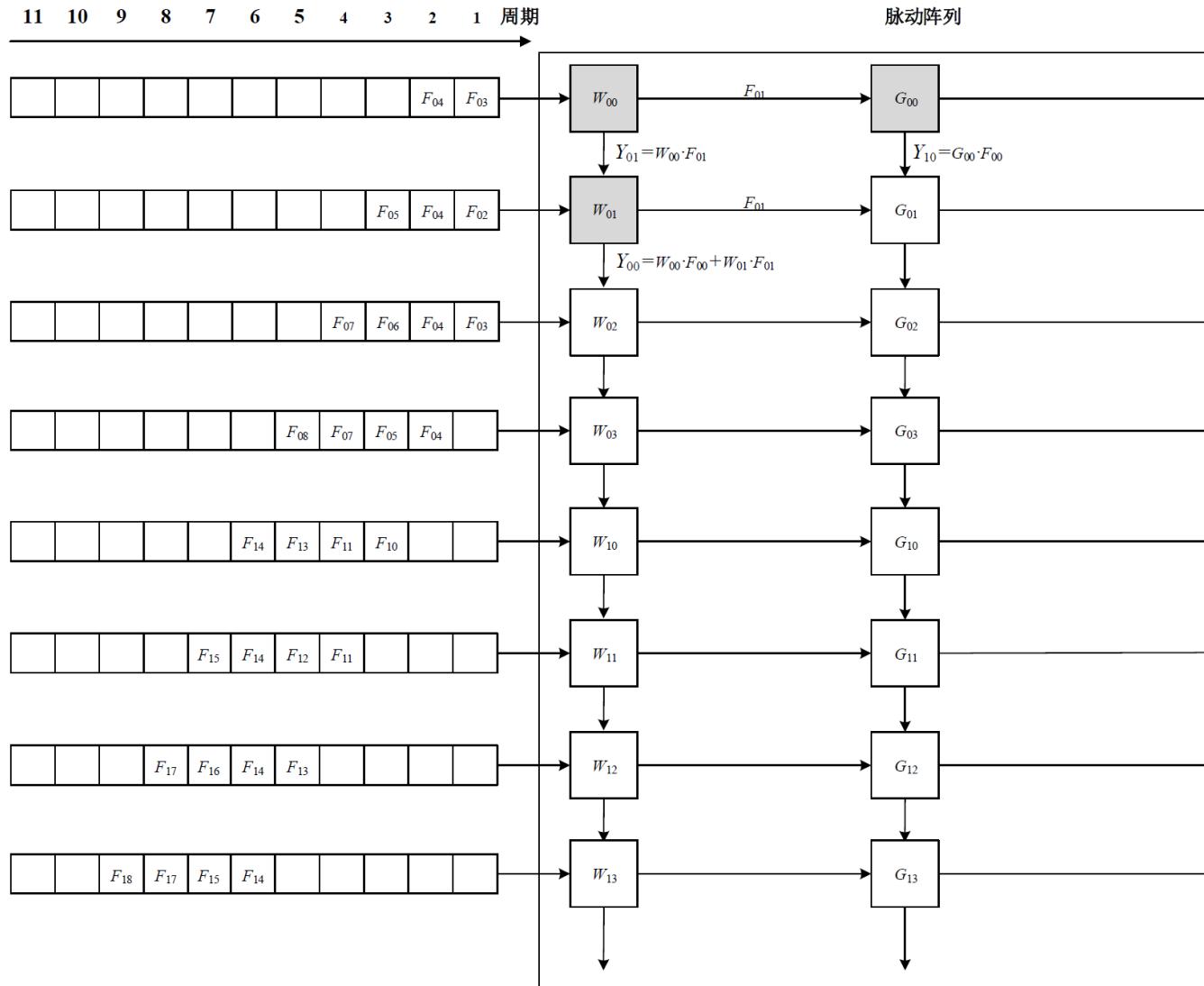


Google TPU (Systolic Array)



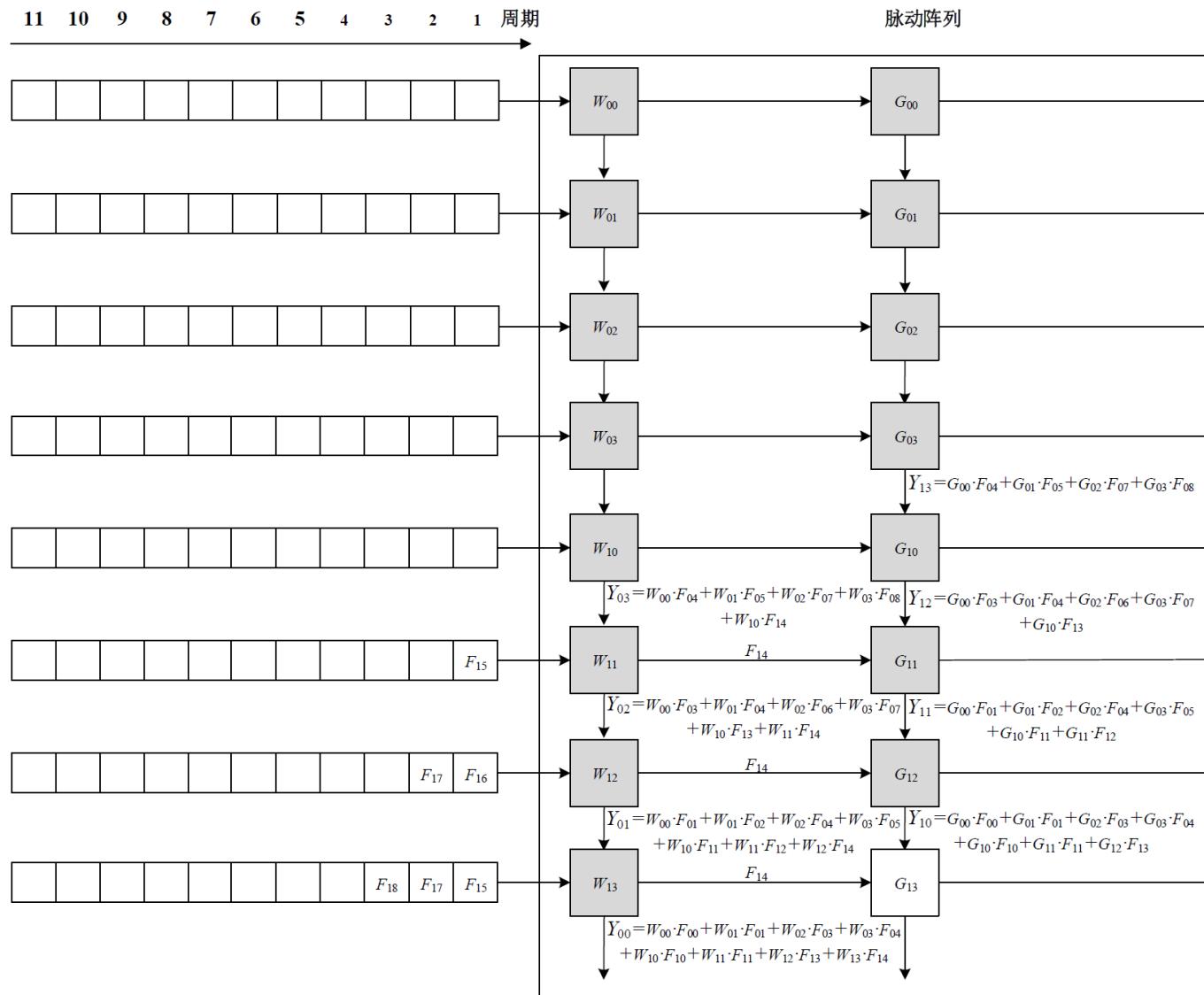


Google TPU (Systolic Array)



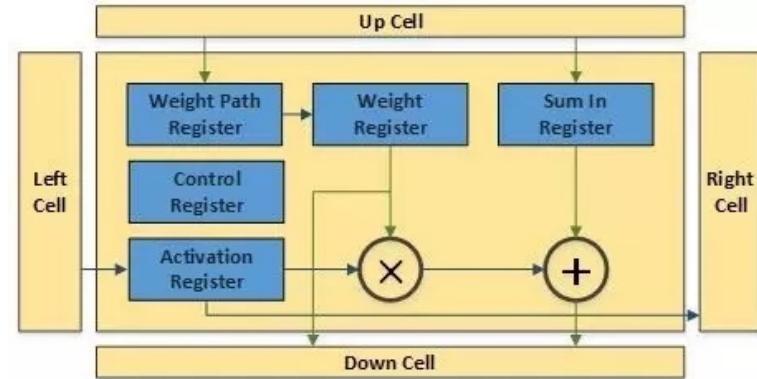
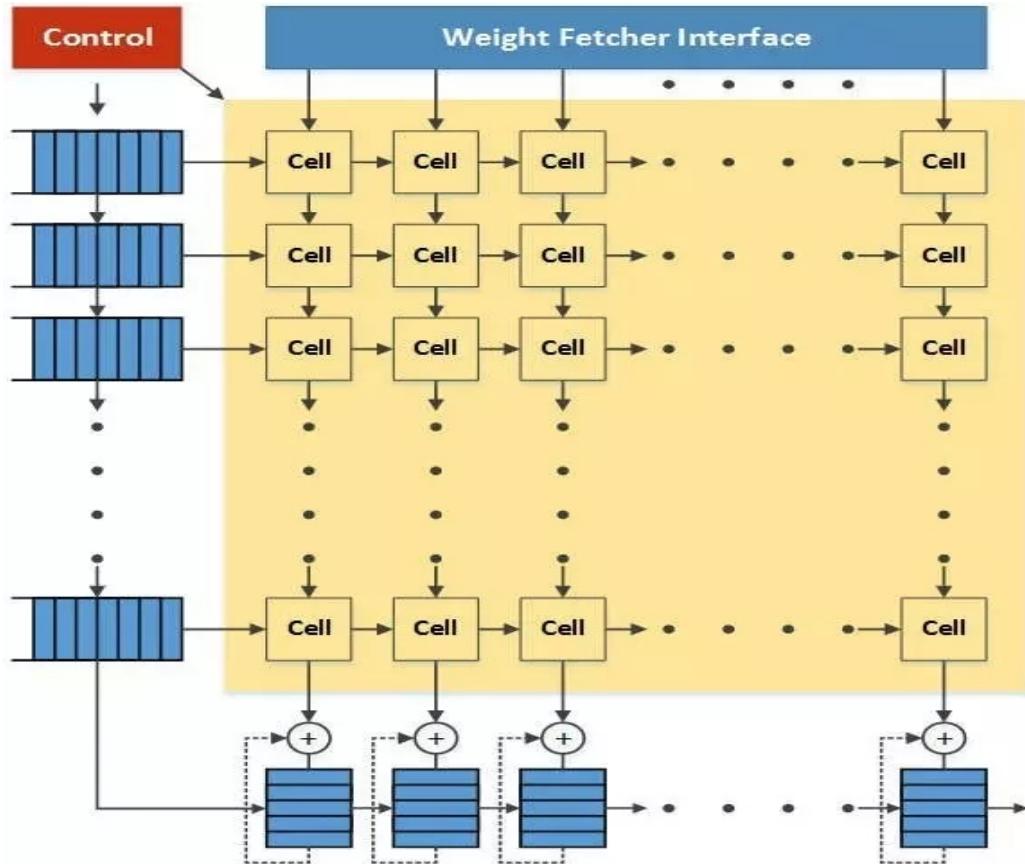


Google TPU (Systolic Array)



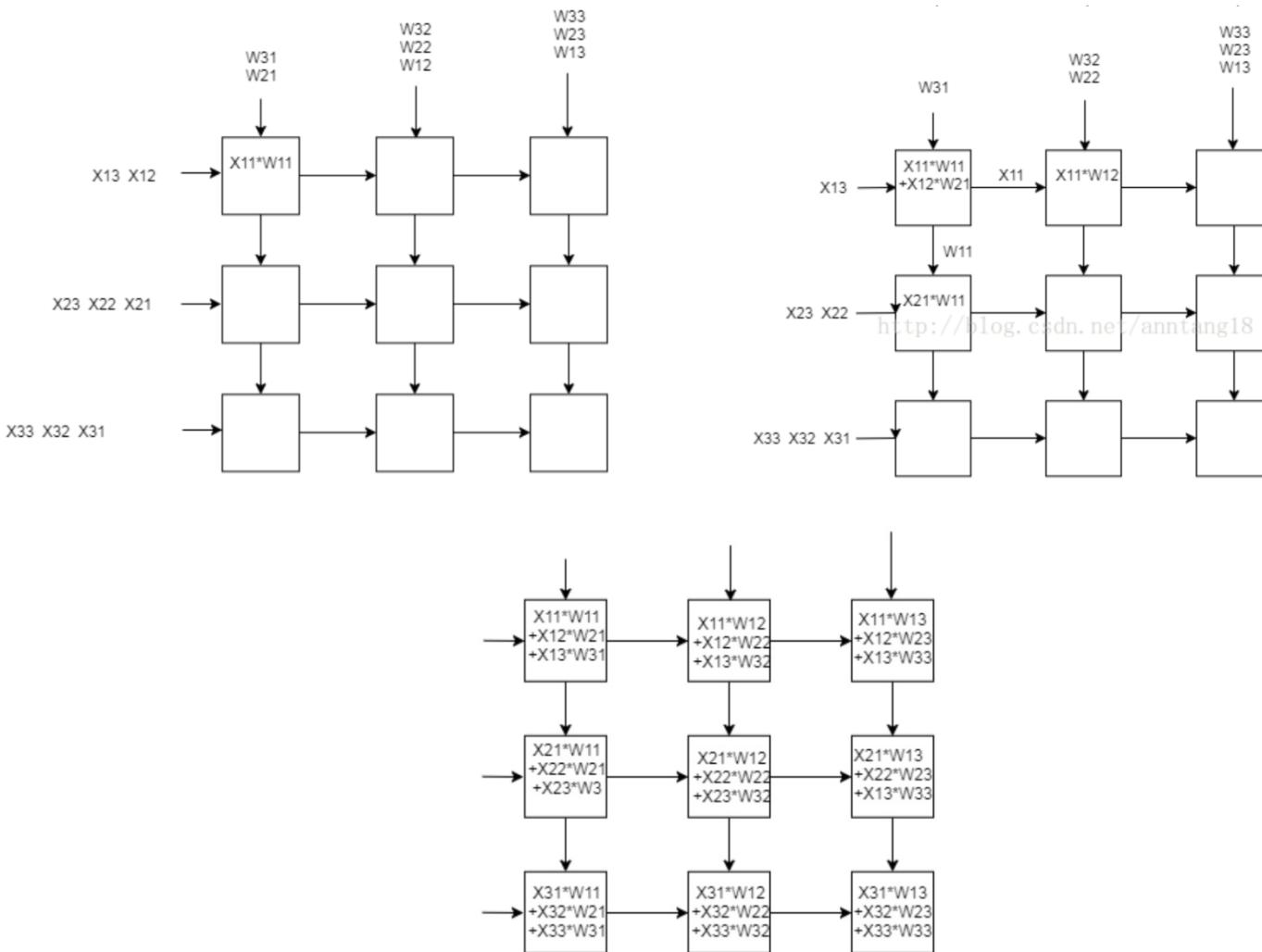


Google TPU (Systolic Array)



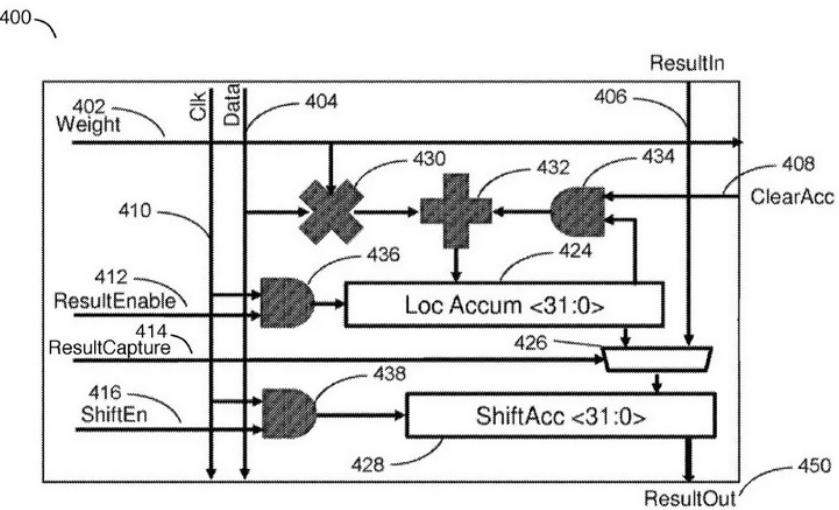
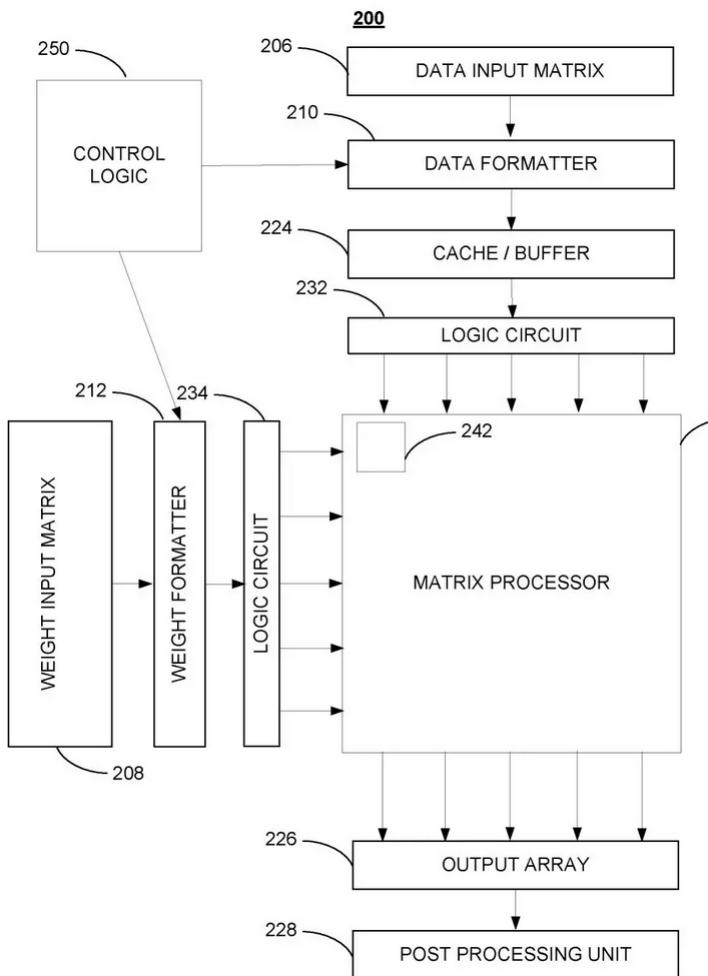


Tesla FSD





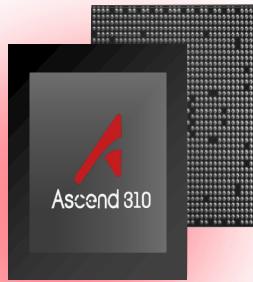
Tesla FSD



昇腾 (Ascend) AI处理器

- NPU (Neural-Network Processing Units, 神经网络处理器) : 在电路层模拟人类神经元和突触，并且用深度学习指令集直接处理大规模的神经元和突触，一条指令完成一组神经元的处理。

NPU的典型代表 —— 华为昇腾AI芯片 (Ascend) 、寒武纪芯片、IBM的TrueNorth。

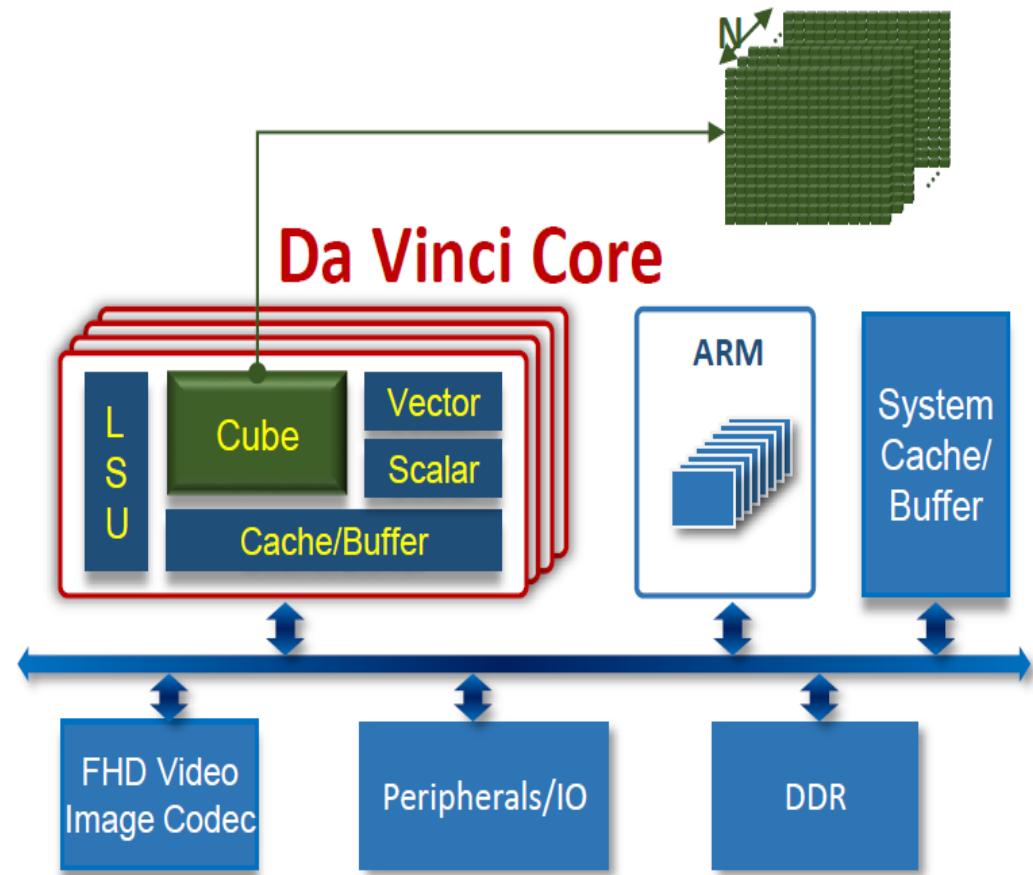


- Ascend-Mini
 - 架构: 达芬奇
 - 半精度 (FP16): 8 Tera-FLOPS
 - 整数精度 (INT8) : 16 Tera-OPS
 - 16 通道 全高清 视频解码器 – H.264/265;
 - 1 通道 全高清 视频编码器 – H.264/265;
 - 最大功耗: 8W
 - 12nm FFC
- Ascend-Max
 - 架构: 达芬奇
 - 半精度 (FP16): 256 Tera-FLOPS
 - 整数精度 (INT8) : 512 Tera-OPS
 - 128 通道 全高清 视频解码器 – H.264/265;
 - 最大功耗: 350W
 - 7nm



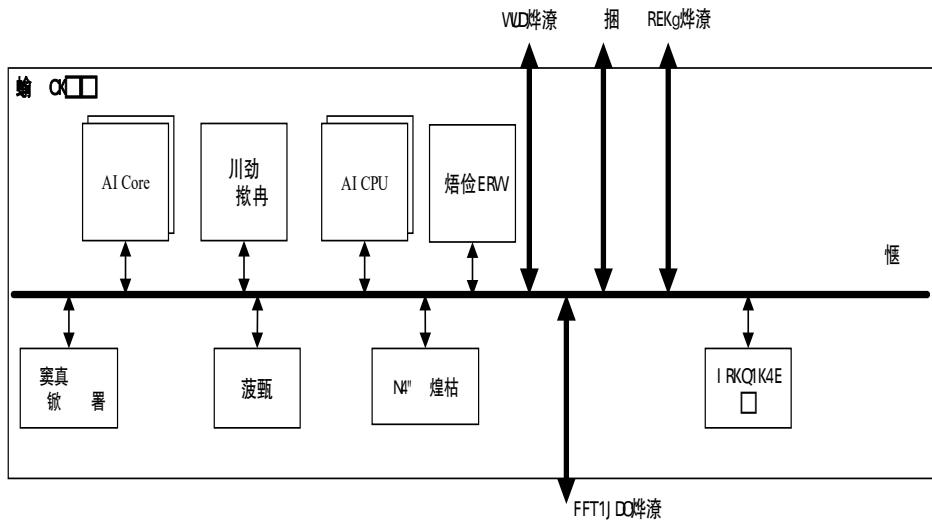
Ascend Processor

SPECIFICATIONS	Description
Architecture	AI co-processor
Performance	Up to 8T @FP16
	Up to 16T@INT8
Codec	16 Channel Decoder – H.264/265 1080P30 1 Channel Encoder
Memory Controller	LPDDR4X
Memory Bandwidth	2*64bit @3733MT/S
System Interface	PCIe3.0 /USB 3.0/GE
Package	15mm*15mm
Max Power	8Tops@4W, 16Tops@8W
Process	12nm FFC



Note: This is typical configuration, high performance and low power sku can be offered based on your requirement.

Ascend310 AI处理器逻辑架构

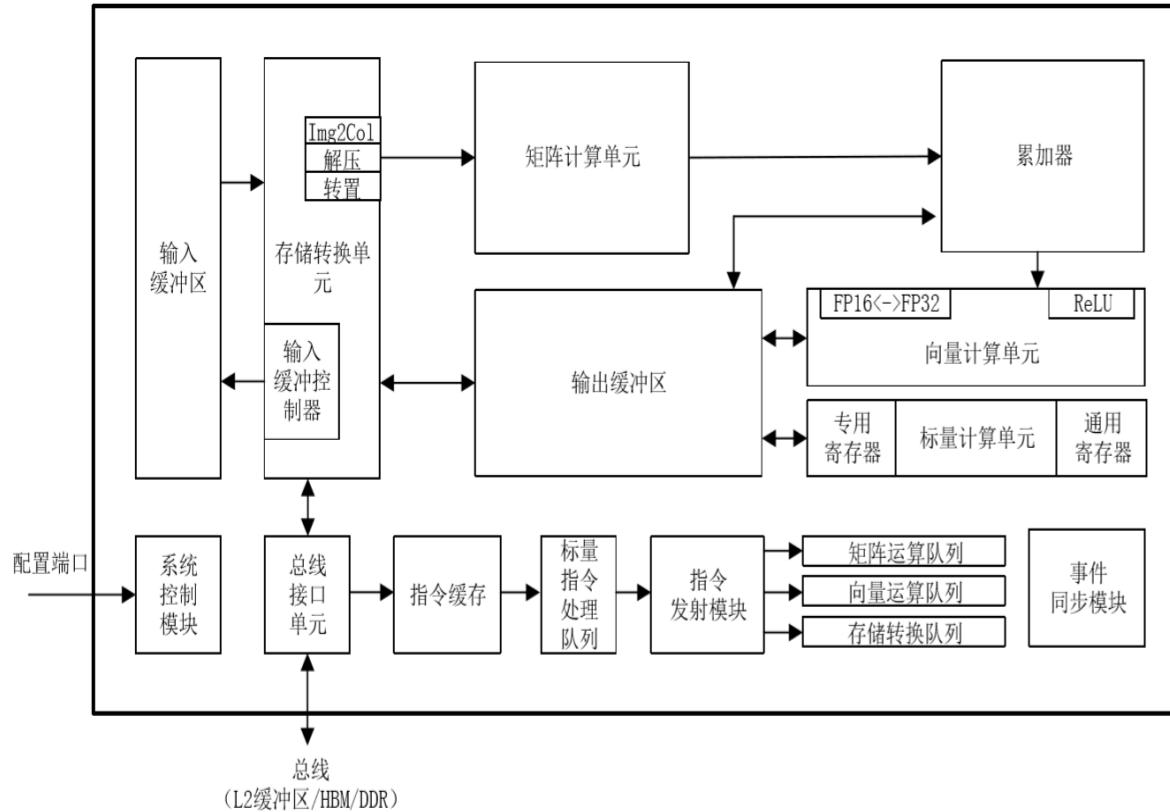


昇腾AI处理器的主要架构组成：

- 芯片系统控制CPU (Control CPU)
- AI计算引擎 (包括AI Core和AI CPU)
- 多层级的片上系统缓存 (Cache) 或缓冲区 (Buffer)
- 数字视觉预处理模块 (Digital Vision Pre-Processing, DVPP) 等

- **AI Core:** 集成了2个AI Core。昇腾AI芯片的计算核心，主要负责执行矩阵、向量、标量计算密集的算子任务，采用达芬奇架构。
- **ARM CPU核心:** 集成了8个A55。其中一部分部署为**AI CPU**，负责执行不适合跑在AI Core上的算子（承担非矩阵类复杂计算）；一部分部署为专用于控制芯片整体运行的**控制CPU**。两类任务占用的CPU核数可由软件根据系统实际运行情况动态分配。此外，还部署了一个专用CPU作为**任务调度器** (Task Scheduler, TS)，以实现计算任务在AI Core上的高效分配和调度；该CPU专门服务于AI Core和AI CPU，不承担任何其他的服务和工作。
- **DVPP:** 数字视觉预处理子系统，完成图像视频的编解码。用于将从网络或终端设备获得的视觉数据，进行预处理以实现格式和精度转换等要求，之后提供给AI计算引擎。
- **Cache & Buffer:** SOC片内有层次化的memory结构，AI core内部有两级memory buffer，SOC片上还有8MB L2 buffer，专用于AI Core、AI CPU，提供高带宽、低延迟的memory访问。芯片还集成了LPDDR4x控制器，为芯片提供更大容量的DDR内存。

达芬奇架构 (AI Core)



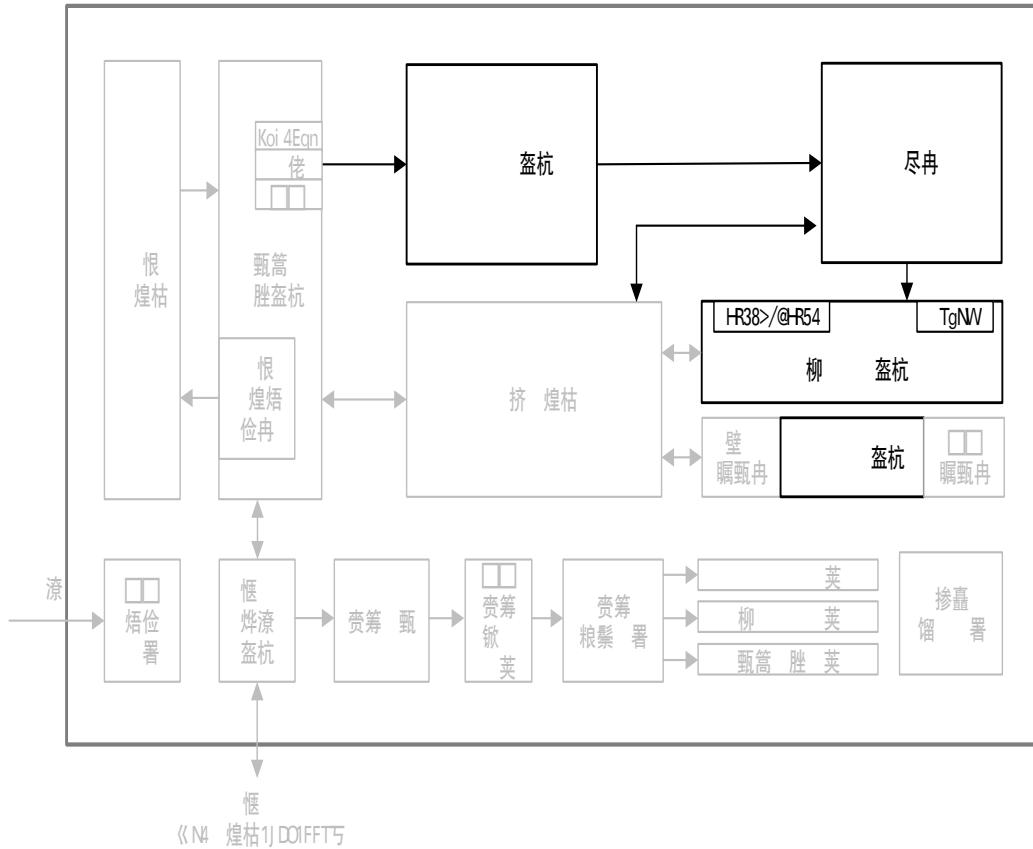
达芬奇架构主要部分：

- **计算单元：**包含三种基础计算资源（矩阵计算单元、向量计算单元、标量计算单元）
- **存储系统：** AI Core的片上存储单元和相应的数据通路构成了存储系统。
- **控制单元：** 整个计算过程提供了指令控制，相当于AI Core的司令部，负责整个AI Core的运行。

达芬奇架构 (AI Core) —— 计算单元

三种基础计算资源：Cube Unit、Vector Unit和Scalar

Unit，分别对应矩阵、向量和标量三种常见的计算模式。



• 矩阵计算单元 (Cube Unit) :

矩阵计算单元和累加器主要完成矩阵相关运算。一拍完成一个fp16的 16×16 与 16×16 矩阵乘 (4096)；如果是int8输入，则一拍完成 16×32 与 32×16 矩阵乘 (8192)；

• 向量计算单元 (Vector Unit) :

实现向量和标量，或双向量之间的计算，功能覆盖各种基本的计算类型和许多定制的计算类型，主要包括 FP16/FP32/Int32/Int8等数据类型的计算；

• 标量计算单元 (Scalar Unit) :

相当于一个微型CPU，控制整个AI Core的运行，完成整个程序的循环控制、分支判断，可以为Cube/Vector提供数据地址和相关参数的计算，以及基本的算术运算。



达芬奇架构 (AI Core) —— 加速原理

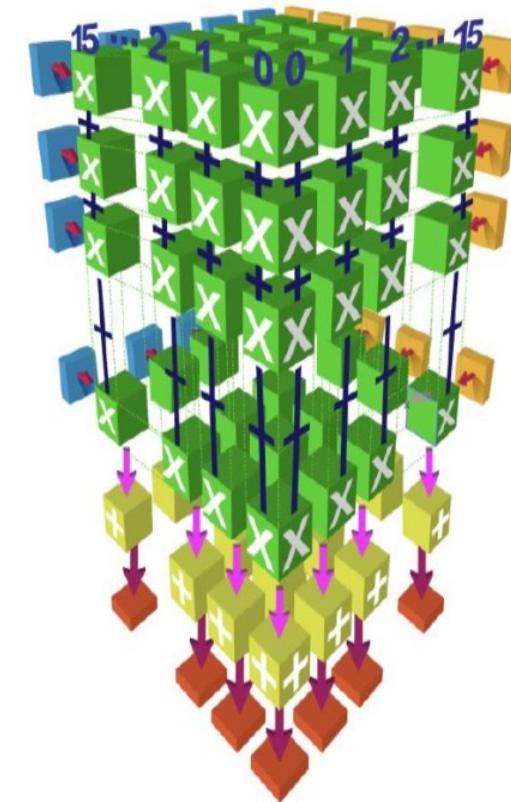
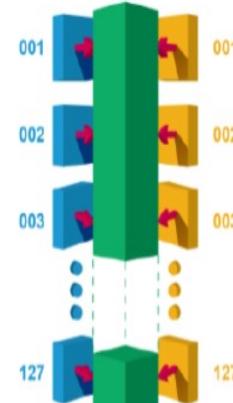
Scalar Unit
Full Flexibility Computation



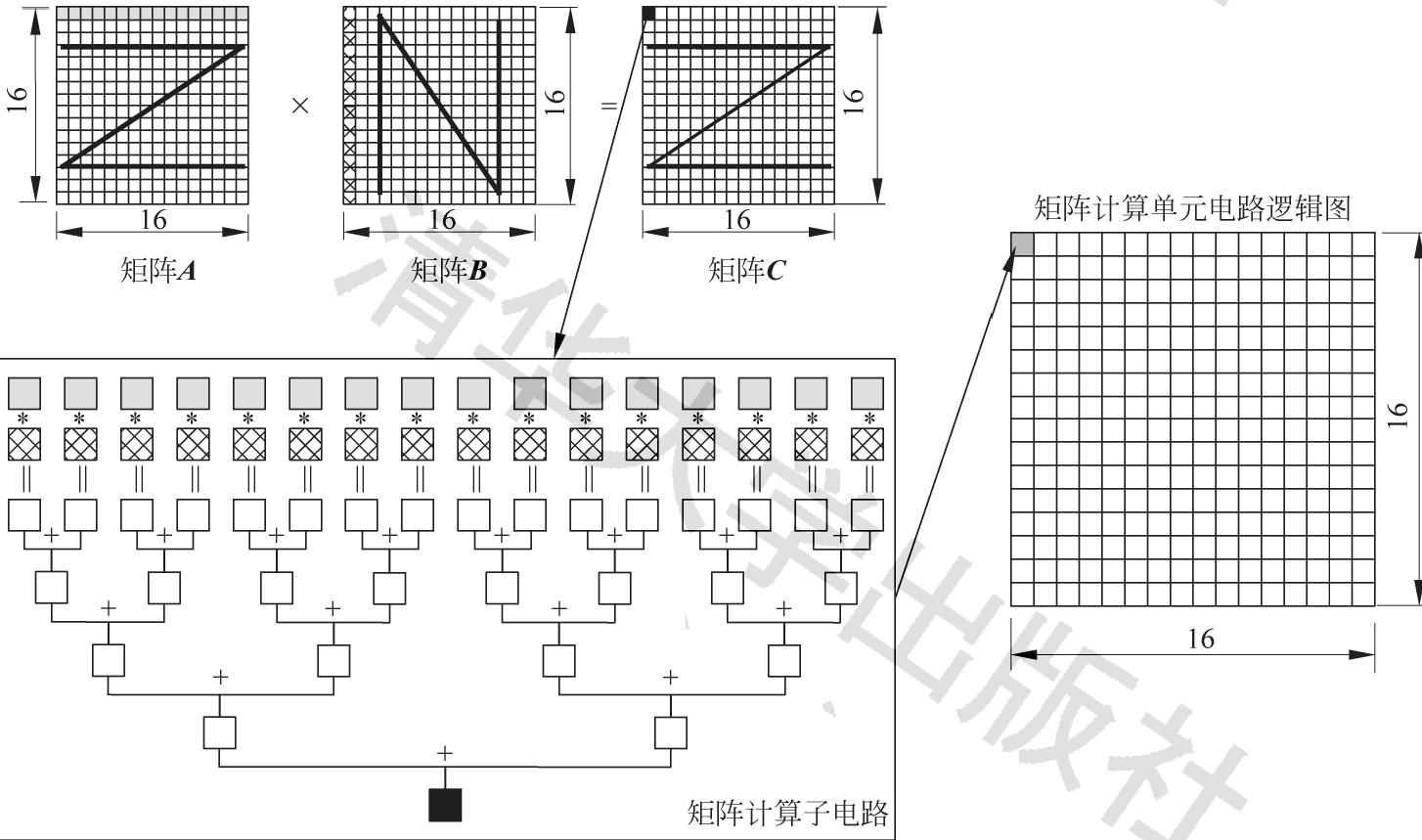
Vector Unit
Rich & Efficient Operations



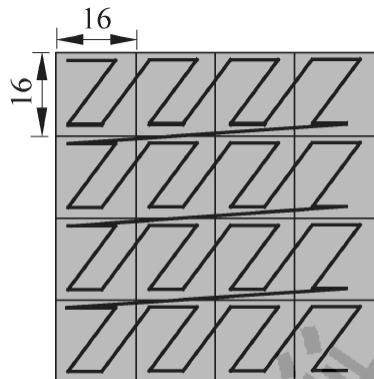
Cube Unit
High Intensity Computation



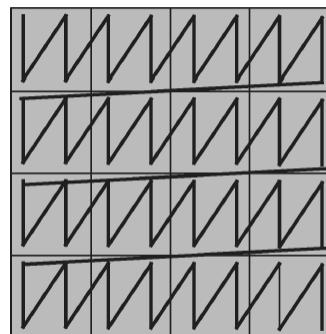
3D-Cube计算



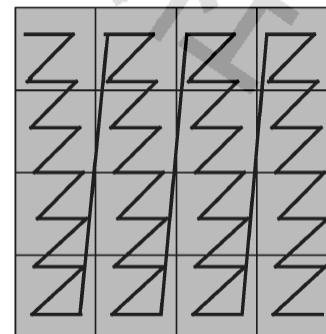
矩阵分块排列方式



矩阵A

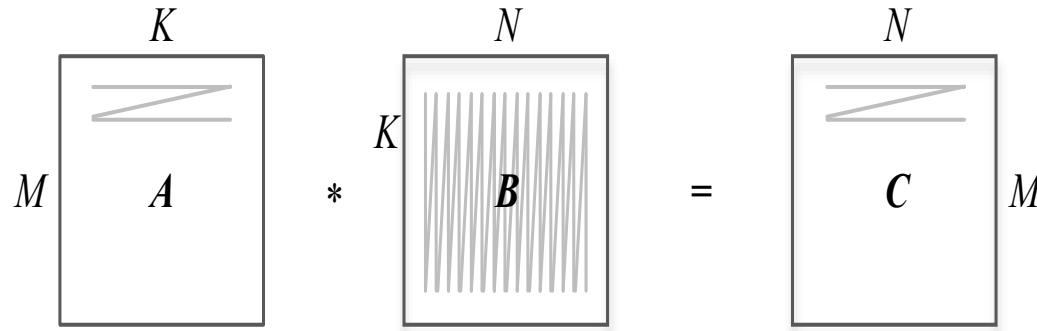


矩阵B



矩阵C

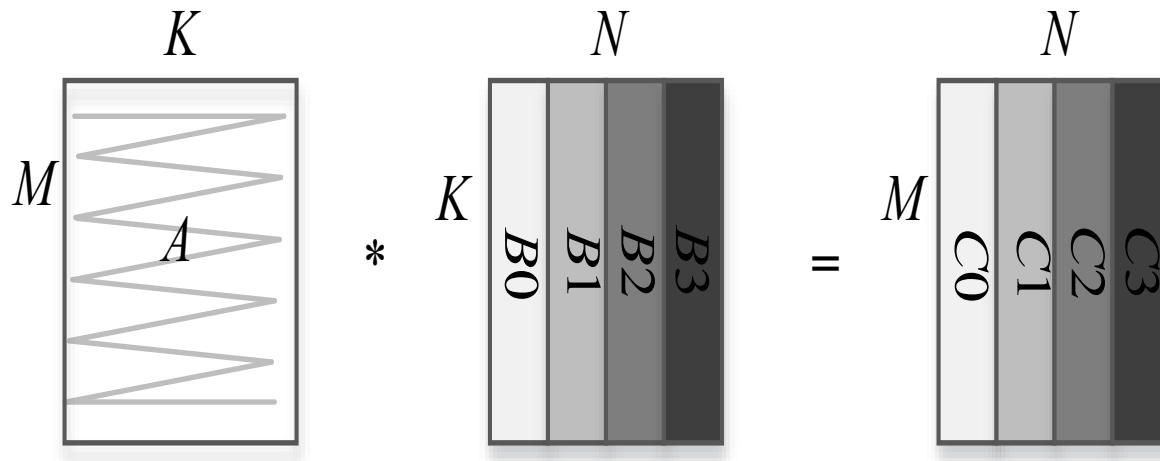
达芬奇架构 (AI Core) —— 矩阵计算单元



- 上图表示一个矩阵A和另一个矩阵B之间的乘法运算 $C = A * B$, 其中M表示矩阵A的行数, K表示矩阵A的列数以及矩阵B的行数, N表示矩阵B的列数。

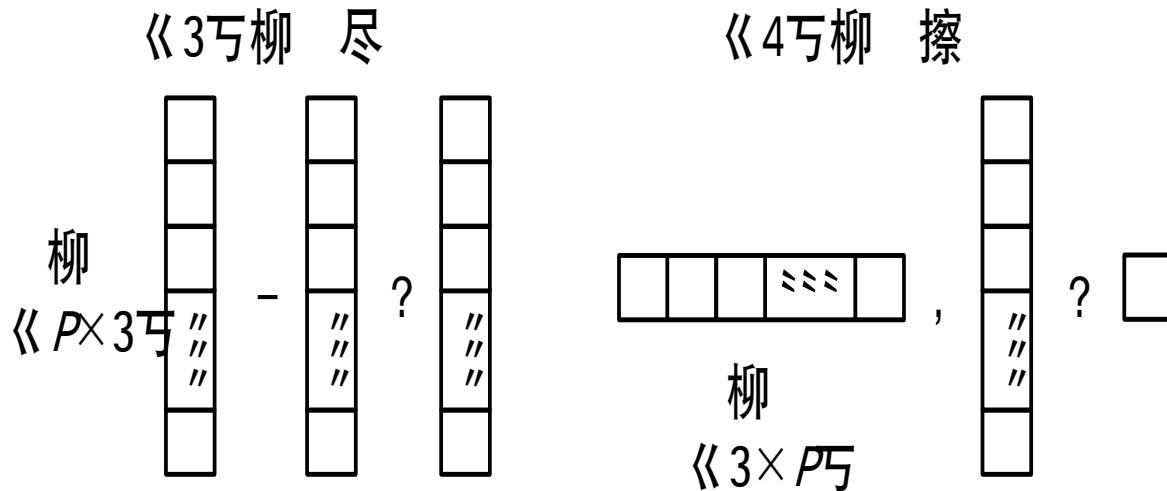
```
➤ for (m=0; m<M, m++)  
➤   for (n=0; n<N, n++)  
➤     for (k=0; k<K, k++)  
➤       C[m][n] += A[m][k]*B[k][n]
```

达芬奇架构 (AI Core) —— 矩阵分块计算



- 一般在矩阵较大时，由于芯片上计算和存储资源有限，往往需要对矩阵进行分块平铺处理（Tiling）。受限于片上缓存的容量，当一次难以装下整个矩阵B时，可以将矩阵B划分成为B0、B1、B2和B3等多个子矩阵。而每一个子矩阵的大小都可以适合一次性存储到芯片上的缓存中并与矩阵A进行计算从而得到结果子矩阵。这样做的目的是充分利用数据的局部性原理，尽可能的把缓存中的子矩阵数据重复使用完毕并得到所有相关的子矩阵结果后再读入新的子矩阵开始新的周期。如此往复可以依次将所有的子矩阵都一一搬运到缓存中，并完成整个矩阵计算的全过程，最终得到结果矩阵C。

达芬奇架构 (AI Core) —— 向量计算单元



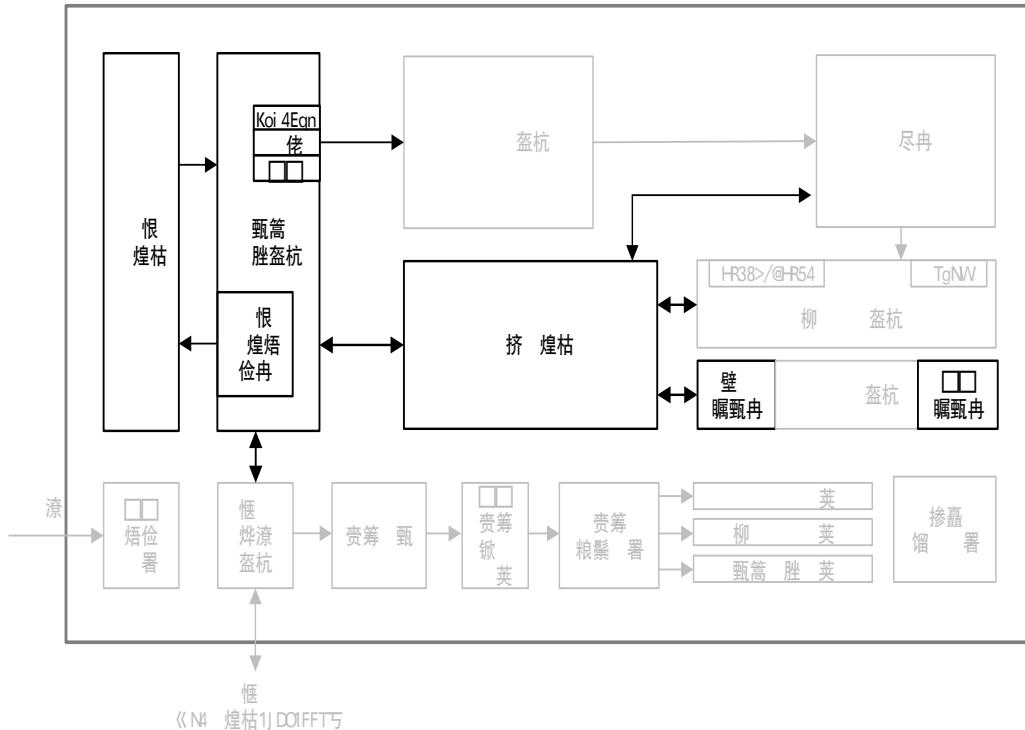
- AI Core中的向量计算单元主要负责完成和向量相关的运算，能够实现向量和标量，或双向量之间的计算，功能覆盖各种基本和多种定制的计算类型，主要包括FP32、FP16、INT32和INT8等数据类型的计算。
- 如上图所示，向量计算单元可以快速完成两个FP16类型的向量相加或者相乘。向量计算单元的源操作数和目的操作数通常都保存在输出缓冲器中。对向量计算单元而言，输入的数据可以不连续，这取决于输入数据的寻址模式。

达芬奇架构 (AI Core) —— 标量计算单元

- 标量计算单元负责完成AI Core中与标量相关的运算。它相当于一个微型CPU，控制整个AI Core的运行。标量计算单元可以对程序中的循环进行控制，可以实现分支判断，其结果可以通过在事件同步模块中插入同步符的方式来控制AI Core中其它功能性单元的执行流水。它还为矩阵计算单元或向量计算单元提供数据地址和相关参数的计算，并且能够实现基本的算术运算。其它复杂度较高的标量运算则由专门的AI CPU通过算子完成。
- 在标量计算单元周围配备了多个通用寄存器 (General Purpose Register, GPR) 和专用寄存器 (Special Purpose Register, SPR)。这些通用寄存器可以用于变量或地址的寄存，为算术逻辑运算提供源操作数和存储中间计算结果。专用寄存器的设计是为了支持指令集中一些指令的特殊功能，一般不可以直接访问，只有部分可以通过指令读写。

达芬奇架构 (AI Core) —— 存储系统

AI Core采用了大容量的片上缓冲区设计，通过增大的片上缓存数据量来减少数据从片外存储系统搬运到AI Core中的频次，从而可以降低数据搬运过程中所产生的功耗，有效控制了整体计算的能耗。



- 数据通路：是指AI Core在完成一次计算任务时，数据在AI Core中的流通路径。达芬奇架构数据通路的特点是多进单出，主要是考虑到神经网络在计算过程中，输入的数据种类繁多并且数量巨大，可以通过并行输入的方式来提高数据流入的效率。与此相反，将多种输入数据处理完成后往往只生成输出特征矩阵，数据种类相对单一，单输出的数据通路，可以节约芯片硬件资源。

存储单元和相应的数据通路，构成了AI Core的存储系统。

存储单元由存储控制单元、缓冲区和寄存器组成。

- **存储控制单元：**

通过总线接口直接访问AI Core之外的更低层级的缓存，也可以直通到DDR或HBM直接访问内存。其中还设置了存储转换单元，作为AI Core内部数据通路的传输控制器，负责AI Core内部数据在不同缓冲区之间的读写管理，以及完成一系列的格式转换操作，如补零，Img2Col，转置、解压缩等；

- **输入缓冲区：**

用来暂时保留需要频繁重复使用的数据，不需要每次都通过总线接口到AI Core的外部读取，从而在减少总线上数据访问频次的同时也降低了总线上产生拥堵的风险，达到节省功耗、提高性能的效果；

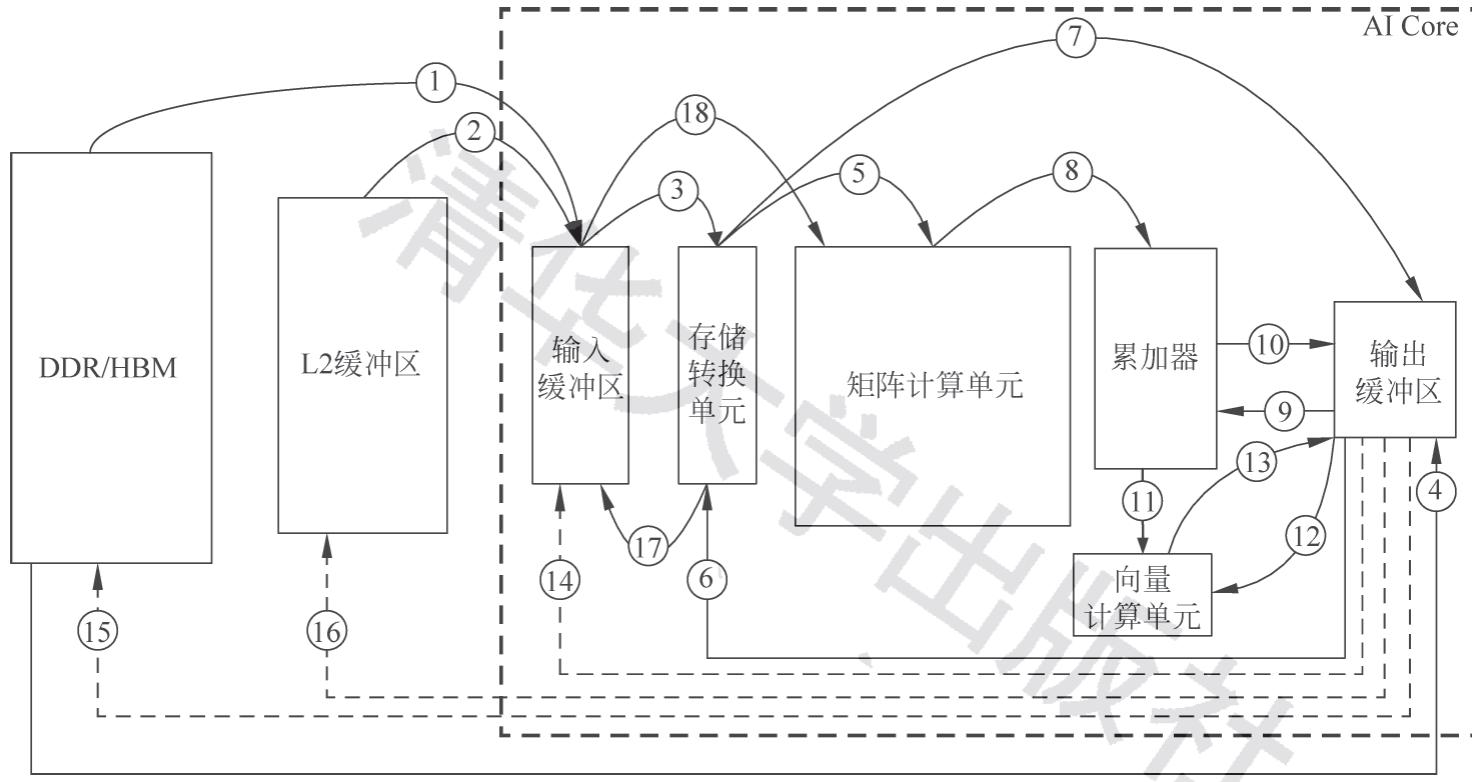
- **输出缓冲区：**

用来存放神经网络中每层计算的中间结果，从而在进入下一层计算时方便地获取数据。相比较通过总线读取数据的带宽低，延迟大，通过输出缓冲区可以大大提升计算效率；

- **寄存器：**

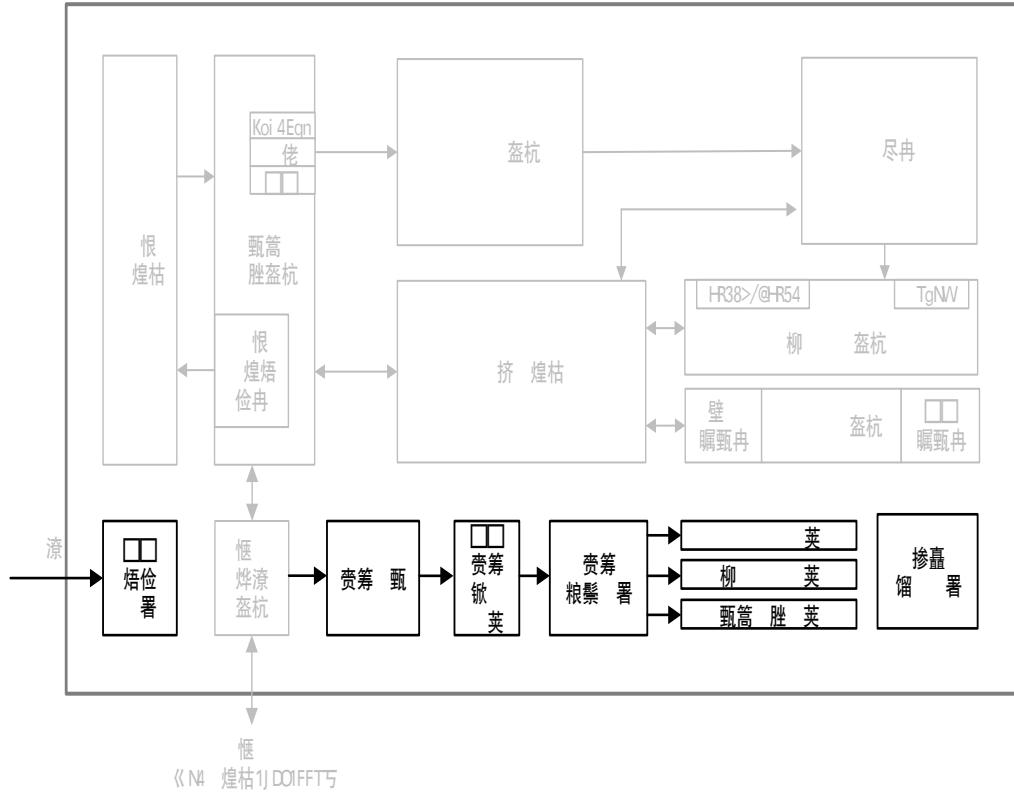
AI Core中的各类寄存器资源主要是标量计算单元在使用。

CNN数据通路



达芬奇架构 (AI Core) —— 控制单元

控制单元主要组成部分为系统控制模块、指令缓存、标量指令处理队列、指令发射模块、矩阵运算队列、向量运算队列、存储转换队列和事件同步模块。



- 系统控制模块:**
控制任务块 (AI Core最小任务粒度) 的执行进程，在任务块执行完成后，系统控制模块会进行中断处理和状态申报。如果执行过程出错，会把执行的错误状态报告给任务调度器；
- 指令缓存:**
在指令执行过程中，可以提前预取后续指令，并一次读入多条指令进入缓存，提升指令执行效率；
- 标量指令处理队列:**
指令被解码后便会被导入标量队列中，实现地址解码与运算控制，这些指令包括矩阵计算指令、向量计算指令以及存储转换指令等；
- 指令发射模块:**
读取标量指令队列中配置好的指令地址和参数解码，然后根据指令类型分别发送到对应的指令执行队列中，而标量指令会驻留在标量指令处理队列中进行后续执行；
- 指令执行队列:**
指令执行队列由矩阵运算队列、向量运算队列和存储转换队列组成，不同的指令进入相应的运算队列，队列中的指令按进入顺序执行；
- 事件同步模块:**
时刻控制每条指令流水线的执行状态，并分析不同流水线的依赖关系，从而解决指令流水线之间的数据依赖和同步的问题。

软件控制下的事件同步方式

矩阵运算队列

向量运算队列

存储转换队列

标量指令处理队列



Case Study

- $N = 10$
- $C_{\text{in}} = 32$
- $C_{\text{out}} = 64$
- $H_{\text{in}} = W_{\text{in}} = H_{\text{out}} = W_{\text{out}} = 28$
- $H_k = W_k = 3$
- $P_h = P_w = 1$
- $S_h = S_w = 1$

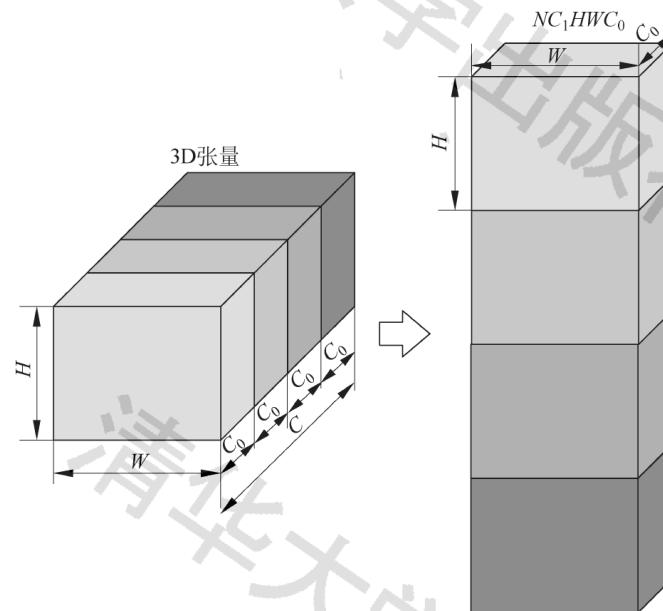
在这种情况下：

- (1) 输入特征图数据为 $[N, H_{\text{in}}, W_{\text{in}}, C_{\text{in}}]$, 即 $[10, 28, 28, 32]$ 。
- (2) 权重数据为 $[C_{\text{out}}, C_{\text{in}} H_k, W_k]$, 即 $[64, 32, 3, 3]$ 。
- (3) 输出特征图数据为 $[N, H_{\text{out}}, W_{\text{out}}, C_{\text{out}}]$, 即 $[10, 28, 28, 64]$ 。

Case Study

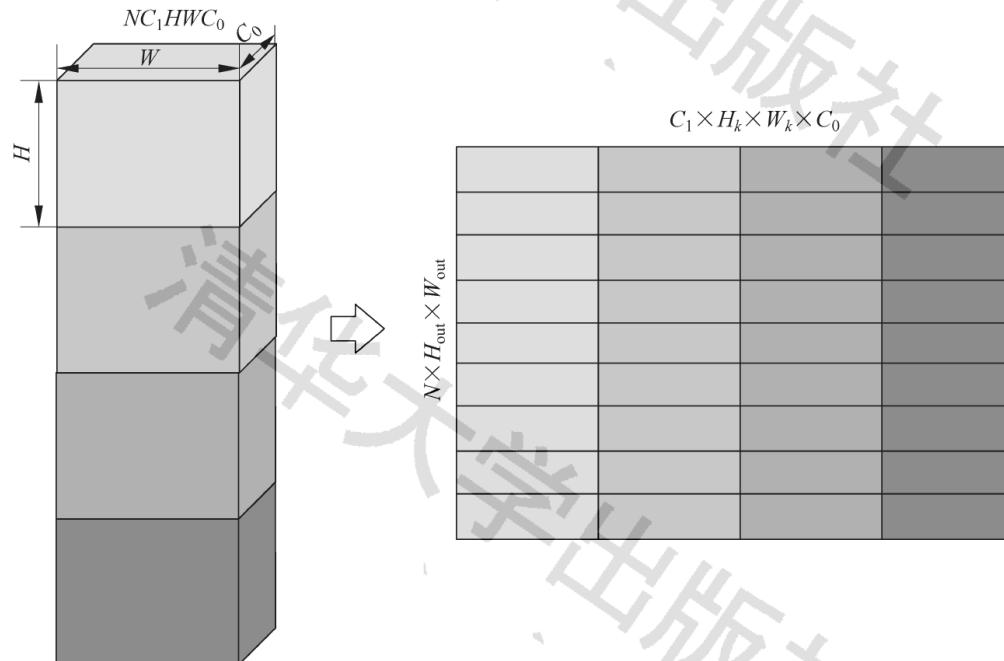
步骤 1 将输入特征图数据 $[N, H_{\text{in}}, W_{\text{in}}, C_{\text{in}}]$ 的 C_{in} 按照 K_{cube} 进行拆分得到 $\left[N, H_{\text{in}}, W_{\text{in}}, \left\lceil \frac{C_{\text{in}}}{K_{\text{cube}}} \right\rceil, K_{\text{cube}}\right]$, 即 $[10, 28, 28, 32]$ 拆分成 $[10, 28, 28, 2, 16]$, 如果 C_{in} 不是 K_{cube} 的倍数, 则需要补零到 K_{cube} 的倍数再进行拆分。

步骤 2 将 $\left[N, H_{\text{in}}, W_{\text{in}}, \left\lceil \frac{C_{\text{in}}}{K_{\text{cube}}} \right\rceil, K_{\text{cube}}\right]$ 进行维度转换, 得到 $\left[N, \left\lceil \frac{C_{\text{in}}}{K_{\text{cube}}} \right\rceil, H_{\text{in}}, W_{\text{in}}, K_{\text{cube}}\right]$, 即 $[10, 2, 28, 28, 16]$ 进一步转换成 $[10, 2, 28, 28, 16]$ 。



Case Study

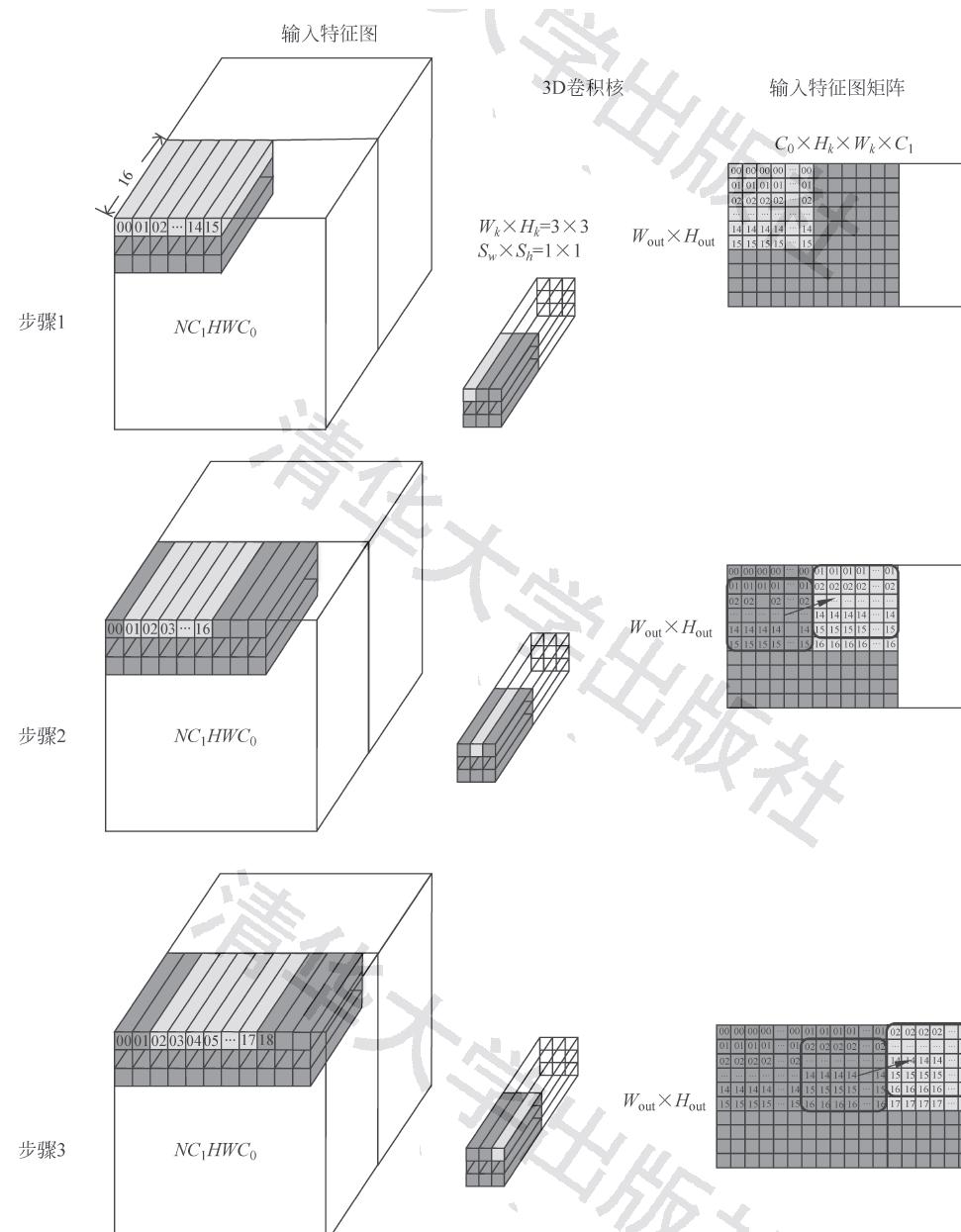
将输入特征图转换成 NC_1HWC_0 的 5D 格式后,根据 Img2Col 算法,需要将数据转换成输入特征图矩阵的 2D 格式,也就是矩阵。根据前文描述的 Img2Col 算法,最终输入特征图矩阵的形状为 $[N, H_{\text{out}}W_{\text{out}}, H_k W_k C_{\text{in}}]$, 其中 $H_{\text{out}}W_{\text{out}}$ 为高, $H_k W_k C_{\text{in}}$ 为宽。如图 6-30 所示,由于昇腾 AI 处理器将 C_{in} 维度进行了拆分,因此最后输入特征图矩阵的形状为 $[N, H_{\text{out}}W_{\text{out}}, \lceil \frac{C_{\text{in}}}{K_{\text{cube}}} \rceil H_k W_k K_{\text{cube}}]$, 即 $[10, 28 \times 28, 2 \times 3 \times 3 \times 16]$, 即 $[10, 784, 288]$ 。



Case Study

经过 Img2Col 转换后,需要进一步将输入特征图矩阵转换成“大 Z 小 Z”的排布格式,因此需要进一步做轴的拆分和转换,最后输入特征图分形格式的形状为 $[N, \lceil \frac{H_{\text{out}}W_{\text{out}}}{M_{\text{cube}}} \rceil, \lceil \frac{C_{\text{in}}}{K_{\text{cube}}} \rceil H_k W_k, M_{\text{cube}}, K_{\text{cube}}]$, 其中 $H_{\text{out}}W_{\text{out}}$ 填充到 M_{cube} 的倍数再进行拆分,带入具体的数据,就是 $[10, 49, 18, 16, 16]$ 。可以看出 $[16, 16]$ 分别为小矩阵的高和宽,按照行优先排列,因此称为“小 Z”排布, $[49, 18]$ 分别为输入特征图矩阵分形格式的高和宽,按照行优先排列,因此称为“大 Z”排布。

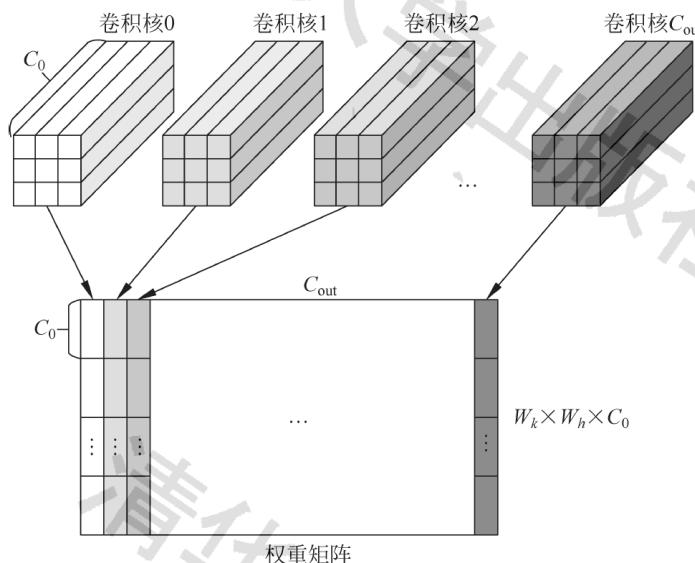
Case Study



Case Study

权重数据的形状为 $[C_{\text{out}}, C_{\text{in}}, H_k, H_w]$, 即 $[64, 32, 3, 3]$, 转换成 5D 格式后, 形状为 $\left[C_{\text{out}}, \left\lceil \frac{C_{\text{in}}}{K_{\text{cube}}} \right\rceil, H_k, H_w, K_{\text{cube}}\right]$, 即 $[64, 2, 3, 3, 16]$ 。

由于数据转换离线完成, 因此可以将 5D 转 2D 和“大 Z 小 N”转换分别进行。如图 6-32 所示, 需要将权重数据转换成权重矩阵, 实现相对比较简单。将每一个通道数为 K_{cube} (C_0) 的卷积核数据展开成一个列向量, 多个卷积核的列向量横向拼接, 便可以得到权重矩阵的一部分, 形状为 $[H_k H_w K_{\text{cube}}, C_{\text{out}}]$, 即 $[144, 64]$ 。之后, 在权重矩阵的高度方向按照 $\left\lceil \frac{C_{\text{in}}}{K_{\text{cube}}} \right\rceil = 2$ 维度重复上述操作, 既可以得到最后的权重矩阵 $\left[\left\lceil \frac{C_{\text{in}}}{K_{\text{cube}}} \right\rceil H_k H_w K_{\text{cube}}, C_{\text{out}}\right]$, 即 $[288, 64]$ 。



Case Study

经过 Img2Col 转换后,需要进一步将权重矩阵转换成“大 Z 小 N”的排布格式,因此需要进一步做轴的拆分和转换,最后权重矩阵分形格式的形状为 $\left[\left[\frac{C_{in}}{K_{cube}}\right] H_k H_w, \left[\frac{C_{out}}{N_{cube}}\right], N_{cube}, K_{cube}\right]$,其中 C_{out} 需要填充到 N_{cube} 再进行拆分,带入具体的数据,就是 $[18,4,16,16]$ 。可以看出, $[16,16]$ 分别是小矩阵的宽和高,按照列优先进行排布,因此称为“小 N”排布; $[18,4]$ 分别是权重矩阵分形格式的高和宽,按照行优先进行排布,因此称为“大 Z”排布。

Case Study

```
fp16 A[490][18][16][16], B[18][4][16][16];  
fp32 C[4][490][16][16];
```

```
for (int k = 0; k < 18; k++) {           // 维度 K  
    for (int n = 0; n < 4; n++) {         // 维度 N  
        for (int m = 0; m < 490; m++) {    // 维度 M  
            // 小矩阵的乘加操作  
            C[n][m] += A[m][k] * B[k][n]);  
        }  
    }  
}
```

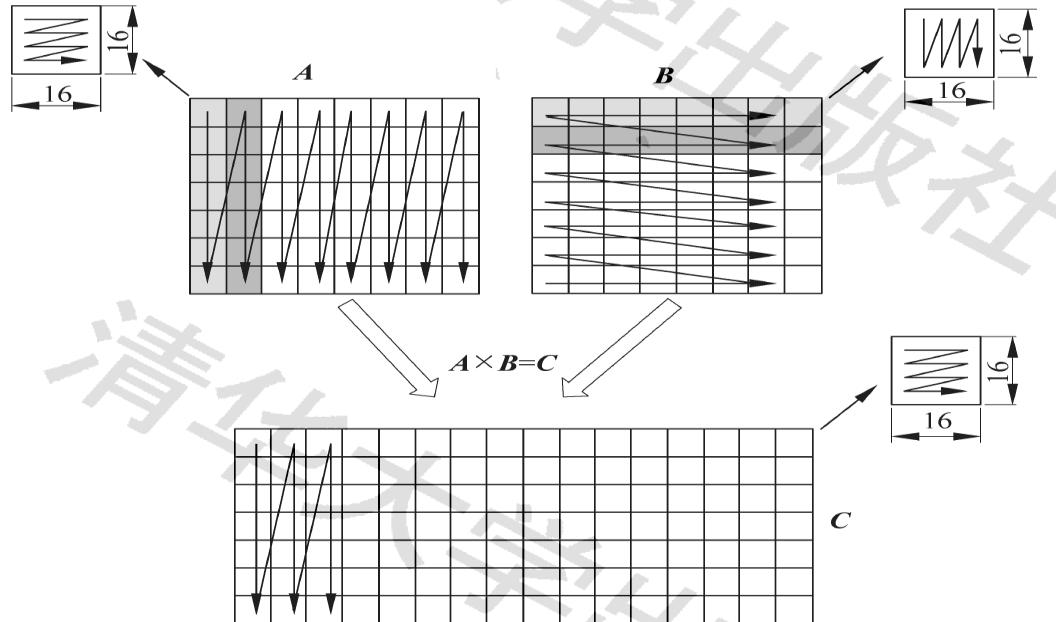


图 6-33 昇腾 AI 处理器中分块矩阵乘法策略

根据以上步骤,最后得到 490×4 个小矩阵,按照“大 N 小 Z”的格式构成输出特征图矩阵,最后的形状为 $[4, 490, 16, 16]$ 。

Case Study

输出特征图矩阵的形状为 $\left[\left\lceil \frac{C_{\text{out}}}{N_{\text{cube}}} \right\rceil, N \left\lceil \frac{H_{\text{out}} W_{\text{out}}}{M_{\text{cube}}} \right\rceil, M_{\text{cube}}, N_{\text{cube}} \right]$, 即 $[4, 490, 16, 16]$ 。

在数据的搬移过程中可以采用特定的读取策略将因为补零造成的冗余去掉, 得到的数据形状为 $\left[\left\lceil \frac{C_{\text{out}}}{N_{\text{cube}}} \right\rceil, N, H_{\text{out}}, W_{\text{out}}, N_{\text{cube}} \right]$, 即 $[4, 10, 28, 28, 16]$; 之后通过维度

转换就可以得到输出特征图数据的 5D 排布格式 $\left[N, \left\lceil \frac{C_{\text{out}}}{N_{\text{cube}}} \right\rceil, H_{\text{out}}, W_{\text{out}}, N_{\text{cube}} \right]$, 即 $[10, 4, 28, 28, 16]$; 在 FP16 的数据类型下, $N_{\text{cube}} = K_{\text{cube}} = 16$, 因此最后得到的输出特征图形状为 $\left[N, \left\lceil \frac{C_{\text{out}}}{K_{\text{cube}}} \right\rceil, H_{\text{out}}, W_{\text{out}}, K_{\text{cube}} \right]$, 即 $[10, 4, 28, 28, 16]$, 为昇腾 AI 处理器规定的 NC_1HWC_0 的 5D 格式。