本页不打印

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 开发/优化者 | 时间 | 审核人 | 开发类型（新开发/优化） | 更新说明 |
| 舒晓东 | 2020.08.01 |  | 新开发 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 课程编码 | 适用产品 | 产品版本 | 课程版本ISSUE |
|  | 智能芯片原理与应用 |  | 1.0 |

修订记录

智能芯片原理与应用



华为技术有限公司

目录

[1 昇腾芯片硬件架构 2](#_Toc48520437)

[1.1 芯片总览 2](#_Toc48520438)

[1.2 达芬奇架构 3](#_Toc48520439)

[1.2.1 达芬奇架构总览 3](#_Toc48520440)

[1.2.2 计算单元 5](#_Toc48520441)

[1.2.3 存储系统 10](#_Toc48520442)

[1.2.4 控制单元 14](#_Toc48520443)

[2 昇腾软件架构 16](#_Toc48520444)

[2.1 昇腾AI软件 16](#_Toc48520445)

[2.1.2 ACL编程开发 17](#_Toc48520446)

[2.2 神经网络软件流 17](#_Toc48520447)

# 昇腾芯片硬件架构

为了满足当今飞速发展的深度神经网络对芯片算力的需求，华为公司于2018年推出了昇腾系列AI处理器，可以对整型数或浮点数提供强大高效的乘加计算力。由于昇腾AI芯片具有强大的算力并且在硬件体系结构上对于深度神经网络进行了特殊的优化，从而使之能以极高的效率完成目前主流深度神经网络的前向计算，因此在智能终端等领域拥有广阔的应用前景。

## 芯片总览

USB接口 网卡 PCIe接口

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **昇腾AI芯片**  AI Core | 任务  调度器 | | AI CPU | 控制CPU |  | |  | 总线 |
|  | | | | | | | | |
|  | | | | | | | | |
| 数字视觉预处理模块 | | 主存 |  | L2 缓冲区 |  | GPIO/I2C  等 | | |

DDR/HBM接口

昇腾 AI 芯片逻辑图

昇腾AI芯片本质上是一个片上系统（System on Chip，SoC），如图1-1所示，主要可以应用在和图像、视频、语音、文字处理相关的应用场景。其主要的架构组成部件包括特制的计算单元、大容量的存储单元和相应的控制单元。该芯片大致可以划为：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（Buffer），数字视觉预处理模块（Digital Vision Pre-Processing，DVPP）等。芯片可以采用LPDDR4高速主存控制器接口，价格较低。目前主流SoC芯片的主存一般DDR（Double Data Rate）或 HBM（High Bandwidth Memory）构成，用来存放大量的数据。HBM相对于DDR存储带宽较高，是行业的发展方向。其它通用的外设接口模块包括USB、磁盘、网卡、GPIO、I2C 和电源管理接口等。

当该芯片作为计算服务器的加速卡使用时，会通过 PCIe 总线接口和服务器其它单元实现数据互换。以上所有这些模块通过基于 CHI 协议的片上环形总线相连，实现模块间的数据连接通路并保证数据的共享和一致性。

昇腾 AI 芯片集成了多个 ARM 公司的 CPU 核心，每个核心都有独立的 L1 和 L2 缓存， 所有核心共享一个片上 L3 缓存。集成的 CPU 核心按照功能可以划分为专用于控制芯片整体运行的主控 CPU 和专用于承担非矩阵类复杂计算的 AI 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。

除了CPU之外，该芯片真正的算力担当是采用了达芬奇架构的AI Core。这些AI Core通过特别设计的架构和电路实现了高通量、大算力和低功耗，特别适合处理深度学习中神经网络必须的常用计算如矩阵相乘等。目前该芯片能对整型数（INT8、INT4）或对浮点数（FP16）提供强大的乘加计算力。由于采用了模块化的设计，可以很方便的通过叠加模块的方法提高后续芯片的计算力。

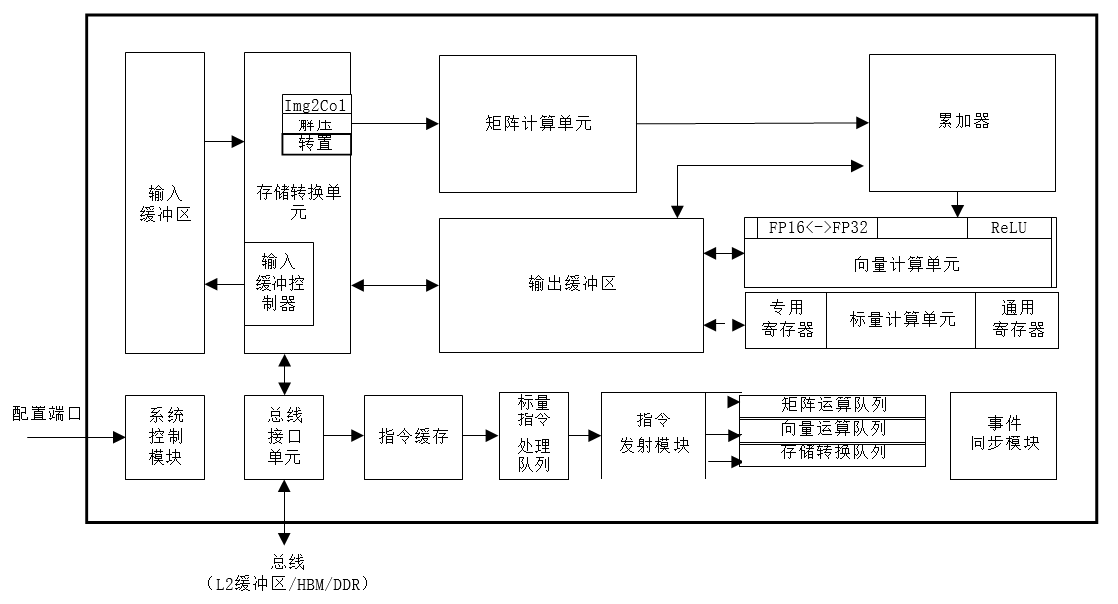
针对深度神经网络参数量大、中间值多的特点，该芯片还特意为AI计算引擎配备了容量为 8MB 的片上缓冲区（On-Chip Buffer），提供高带宽、低延迟、高效率的数据交换和访问。能够快速访问到所需的数据对于提高神经网络算法的整体性能至关重要，同时将大量需要复用的中间数据缓存在片上对于降低系统整体功耗意义重大。为了能够实现计算任务在AI Core上的高效分配和调度，还特意配备了一个专用CPU作为任务调度器（Task Scheduler，TS）。该CPU专门服务于AI Core和 AI CPU，而不承担任何其他的事务和工作。

数字视觉预处理模块主要完成图像视频的编解码，支持4K①分辨率，视频处理，对图像支持JPEG 和PNG等格式的处理。来自主机端存储器或网络的视频和图像数据，在进入昇腾AI芯片的计算引擎处理之前，需要生成满足处理要求的输入格式、分辨率等，因此需要调用数字视觉预处理模块进行预处理以实现格式和精度转换等要求。数字视觉预处理模块主要实现视频解码（Video Decoder，VDEC），视频编码（Video Encoder，VENC），JPEG编解码（JPEG Decoder/Encoder，JPEGD/E），PNG解码（PNG Decoder，PNGD）和视觉预处理（Vision Pre-Processing Core，VPC）等功能。图像预处理可以完成对输入图像的上/下采样、裁剪、色调转换等多种功能。数字视觉预处理模块采用了专用定制电路的方式来实现高效率的图像处理功能，对应于每一种不同的功能都会设计一个相应的硬件电路模块来完成计算工作。在数字视觉预处理模块收到图像视频处理任务后，会读取需要处理的图像视频数据并分发到内部对应的处理模块进行处理，待处理完成后将数据写回到内存中等待后续步骤。

## 达芬奇架构

### 达芬奇架构总览

不同于传统的支持通用计算的CPU和GPU，也不同于专用于某种特定算法的专用芯ASIC，达芬奇架构本质上是为了适应某个特定领域中的常见的应用和算法，通常称之为“特定域架构（Domain Specific Architecture，DSA）”芯片。



AI Core架构图

昇腾AI芯片的计算核心主要由AI Core构成，负责执行标量、向量和张量相关的计算密集型算子。AI Core采用了达芬奇架构，其基本结构如图1-2所示，从控制上可以看成是一个相对简化的现代微处理器的基本架构。它包括了三种基础计算资源：矩阵计算单元

（Cube Unit）、向量计算单元（Vector Unit）和标量计算单元（Scalar Unit）。这三种计算单元分别对应了张量、向量和标量三种常见的计算模式，在实际的计算过程中各司其职，形成了三条独立的执行流水线，在系统软件的统一调度下互相配合达到优化的计算效率。此外在矩阵计算单元和向量计算单元内部还提供了不同精度、不同类型的计算模式。AI Core中的矩阵计算单元目前可以支持INT8、INT4和FP16的计算；向量计算单元目前可以支持FP16 和FP32的计算。

为了配合AI Core中数据的传输和搬运，围绕着三种计算资源还分布式的设置了一系列的片上缓冲区，比如用来放置整体图像特征数据、网络参数以及中间结果的输入缓冲区

（Input Buffer，IB）和输出缓冲区（Output Buffer，OB），以及提供一些临时变量的高速寄存器单元，这些寄存器单元位于各个计算单元中。这些存储资源的设计架构和组织方式不尽相同，但目的都是为了更好的适应不同计算模式下格式、精度和数据排布的需求。这些存储资源和相关联的计算资源相连，或者和总线接口单元（Bus Interface Unit，BIU）相连从而可以获得外部总线上的数据。

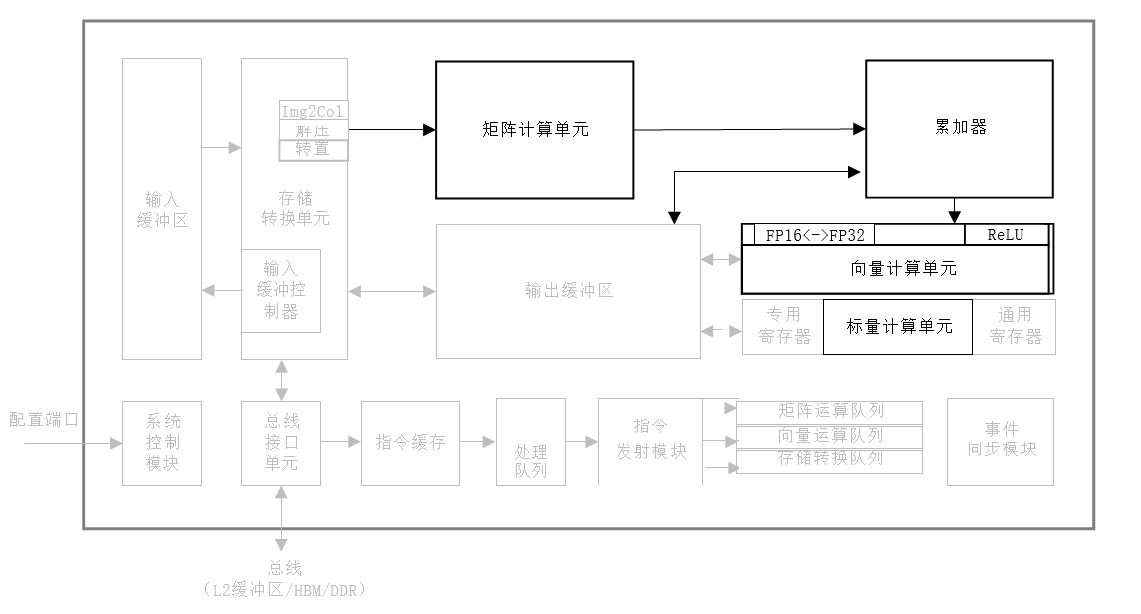
在AI Core中，输入缓冲区之后设置了一个存储转换单元（Memory Transfer Unit，MTE）。这是达芬奇架构的特色之一，主要的目的是为了以极高的效率实现数据格式的转换。比如前面提GPU要通过矩阵计算来实现卷积，首先要通过Img2Col 的方法把输入的网络和特征数据重新以一定的格式排列起来。这一步在GPU当中是通过软件来实现的，效率比较低下。达芬奇架构采用了一个专用的存储转换单元来完成这一过程，将这一步完全固化在硬件电路中，可以在很短的时间之内完成整个转置过程。由于类似转置的计算在深度神经网络中出现的极为频繁，这样定制化电路模块的设计可以提升AI Core的执行效率，从而能够实现不间断的卷积计算。

AI Core中的控制单元主要包括系统控制模块、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。系统控制模块负责指挥和协调 AI Core 的整体运行模式，配置参数和实现功耗控制等。标量指令处理队列主要实现控制指令的译码。当指令被译码并通过指令发射模块顺次发射出去后，根据指令的不同类型，将会分别被发送到矩阵运算队列、向量运算队列和存储转换队列。三个队列中的指令依据先进先出的方式分别输出到矩阵计算单元、向量计算单元和存储转换单元进行相应的计算。不同的指令阵列和计算资源构成了独立的流水线，可以并行执行以提高指令执行效率。如果指令执行过程中出现依赖关系或者有强制的时间先后顺序要求，则可以通过事件同步模块来调整和维护指令的执行顺序。事件同步模块完全由软件控制，在软件编写的过程中可以通过插入同步符的方式来指定每一条流水线的执行时序从而达到调整指令执行顺序的目的。

在AI Core中，存储单元为各个计算单元提供转置过并符合要求的数据，计算单元返回运算的结果给存储单元，控制单元为计算单元和存储单元提供指令控制，三者相互协调合作完成计算任务。

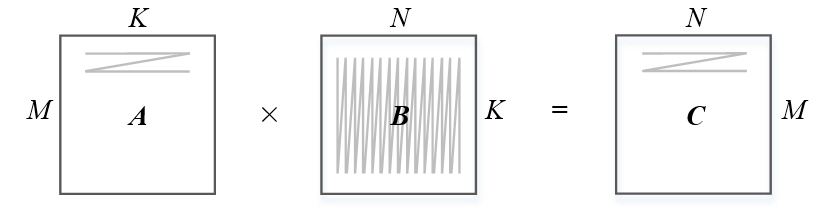
### 计算单元

计算单元是AI Core中提供强大算力的核心单元，相当于AI Core的主力军。AI Core计算单元主要包含矩阵计算单元、向量计算单元、标量计算单元和累加器，如图 1-3 中的加粗所示。矩阵计算单元和累加器主要完成与矩阵相关的运算，向量计算单元负责执行向量运算，标量计算单元主要负责各类型的标量数据运算和程序的流程控制。



计算单元

#### 矩阵计算单元

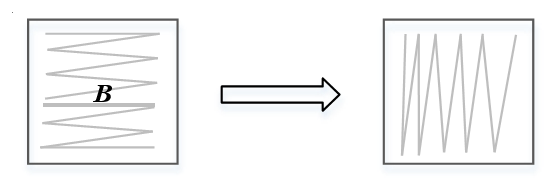


矩阵乘法示意图

由于常见的深度神经网络算法中大量的使用了矩阵计算，达芬奇架构中特意对矩阵计算 进行了深度的优化并定制了相应的矩阵计算单元来支持高吞吐量的矩阵处理。图 1-4 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。

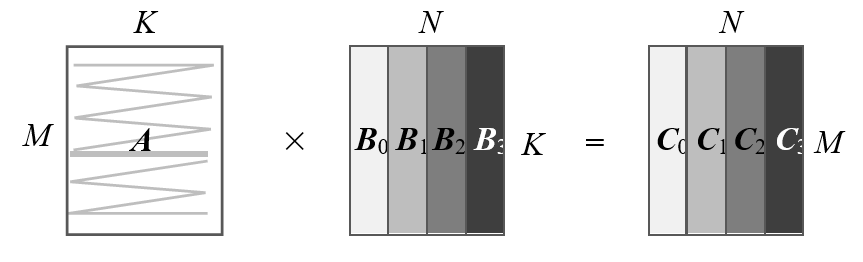
该程序需要用到3个循环进行一次完整的矩阵相乘计算，如果在一个单发射的CPU上执行至少需要 M\*K\*N 个时钟周期才能完成，当矩阵非常庞大时执行过程极为耗时。

在CPU计算过程中，矩阵A是按照行的方式进行扫描，矩阵B以列的方式进行扫描。考虑到典型的矩阵存储方式，无论矩阵A还是矩阵B都会按照行的方式进行存放，也就是所谓的Row-Major的方式。而内存读取的方式是具有极强的数据局部性特征的，也就是说当读取内存中某个数的时候会打开内存中相应的一整行并且把同一行中所有的数都读取出来。这种内存的读取方式对矩阵A是非常高效的，但是对于矩阵B的读取却显得非常不友好，因为代码中矩阵B是需要一列一列读取的。为此需要将矩阵B的存储方式转成按列存储，也就是所谓的Column-Major，如图 1-5 所示，这样才能够符合内存读取的高效率模式。因此，在矩阵计算中往往通过改变某个矩阵的存储方式来提升矩阵计算的效率。



矩阵B存储方式

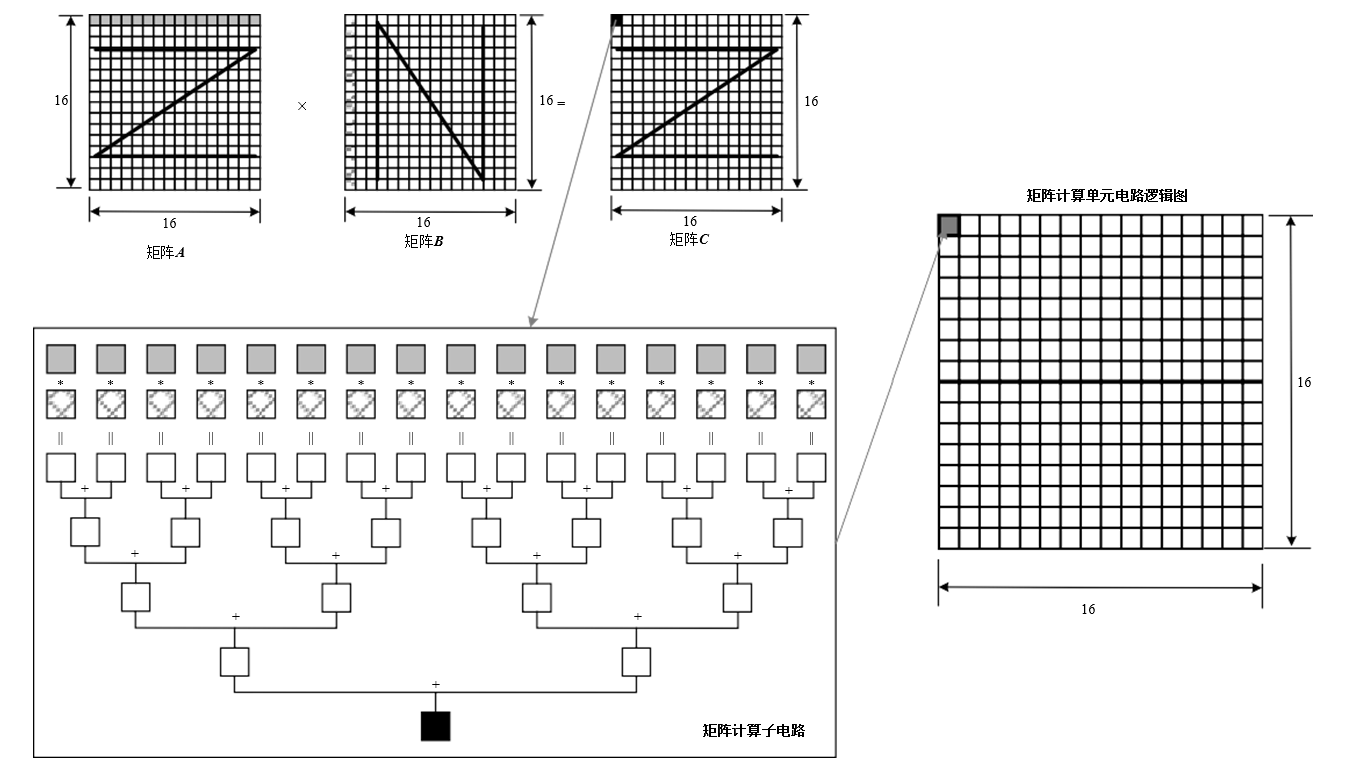
一般在矩阵较大时，由于芯片上计算和存储资源有限，往往需要对矩阵进行分块平铺处理（Tiling），如图1-6所示。受限于片上缓存的容量，当一次难以装下整个矩阵B时，可以将矩阵B划分成为 B0、B1、B2 和、B3等多个子矩阵。而每一个子矩阵的大小都可以适合一次性存储到芯片上的缓存中并与矩阵 A 进行计算从而得到结果子矩阵。这样做的目的是充分利用数据的局部性原理，尽可能的把缓存中的子矩阵数据重复使用完毕并得到所有相关的子矩阵结果后再读入新的子矩阵开始新的周期。如此往复可以依次将所有的子矩阵都一一搬运到缓存中，并完成整个矩阵计算的全过程，最终得到结果矩阵C。分块的优点是充分利用了缓存的容量，并最大程度利用了数据计算过程中的局部性特征，可以高效实现大规模的矩阵乘法计算，是一种常见的优化手段。



矩阵分块计算

在深度神经网络中实现计算卷积过程，关键的步骤是将卷积运算转化为矩阵运算。在CPU 中大规模的矩阵计算往往成为性能瓶颈，而矩阵计算在深度学习算法中又极为重要。为了解决这个矛盾， GPU 采用通用矩阵乘法（GEMM）的方法来实现矩阵乘法。例如要实现一个 16\*16 矩阵与另一个 16\*16 矩阵的乘法，需要安排 256 个并行的线程，并且每一个线程都可以独立计算完成结果矩阵中的一个输出点。假设每一个线程在一个时钟周期内可以完成一次乘加运算，则GPU完成整个矩阵计算需要16个时钟周期，这个延时是GPU无法避免的瓶颈。而昇腾AI芯片针对这个问题做了深度的优化。因此AI Core对矩阵乘法运算的高效性为昇腾AI芯片作为深度神经网络的加速器提供了强大的性能保障。

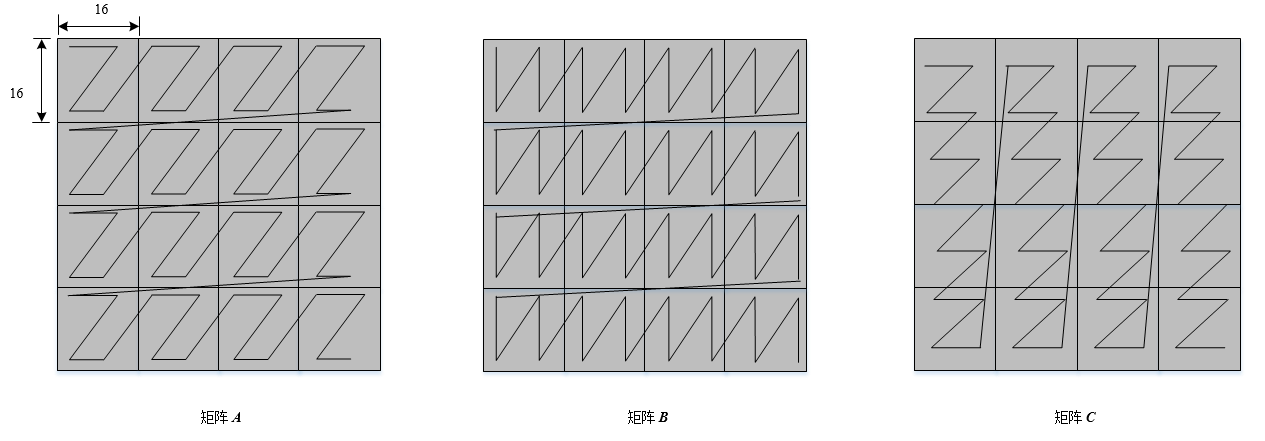
达芬奇架构在AI Core中特意设计了矩阵计算单元作为昇腾AI芯片的核心计算模块， 意图高效解决矩阵计算的瓶颈问题。矩阵计算单元提供强大的并行乘加计算能力，使得AI Core能够高速处理矩阵计算问题。通过精巧设计的定制电路和极致的后端优化手段，矩阵计算单元可以用一条指令完成两个 16\*16 矩阵的相乘运算（标记为 16^3，也是 Cube 这一名称的来历），等同于在极短时间内进行了 16^3=4096 个乘加运算，并且可以实现 FP16 的运算精度。如图 3-7 所示，矩阵计算单元在完成 A\*B=C 的矩阵运算时，会事先将矩阵 A 按行存放在输入缓冲区中，同时将矩阵B按列存放在输入缓冲区中，通过矩阵计算单元计算后得到的结果矩阵 C 按行存放在输出缓冲区中。在矩阵相乘运算中，如图 1-7 所示，矩阵 C 的第一元素由矩阵 A 的第一行的16个元素和矩阵 B 的第一列的16个元素由矩阵计算单元子电路进行16次乘法和15次加法运算得出。矩阵计算单元中共有256个矩阵计算子电路组成，可以由一条指令并行完成矩阵 C 的 256 个元素计算。



矩阵计算单元计算示意图

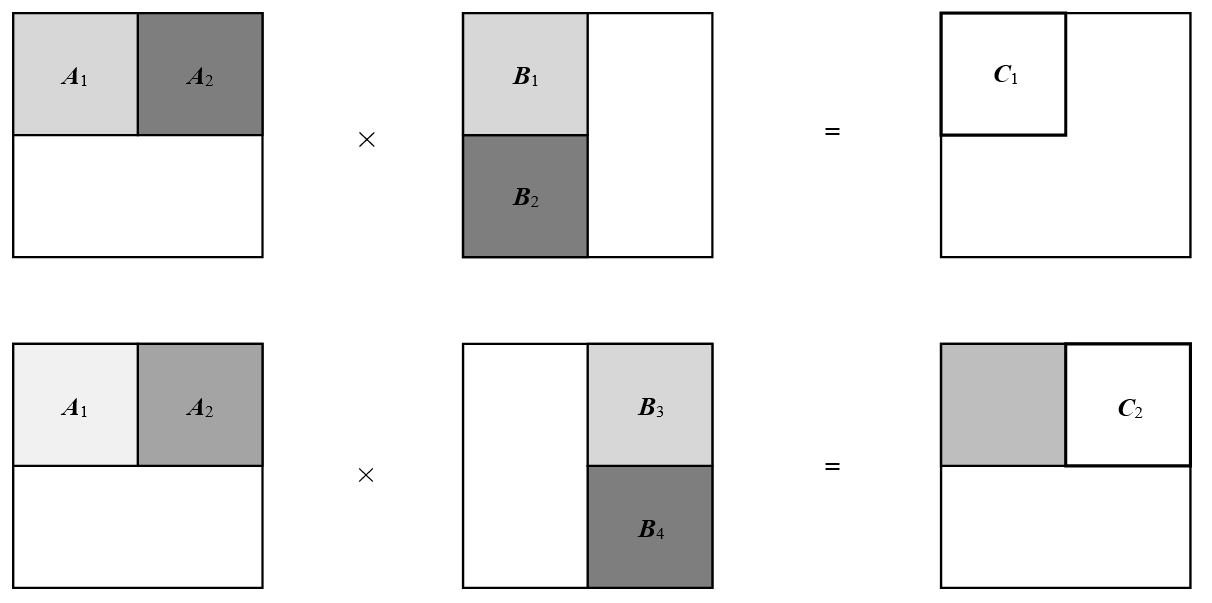
在有关矩阵的处理上，通常在进行完一次矩阵乘法后还需要和上一次的结果进行累加， 以实现类似 C=A\*B+C 的运算。矩阵计算单元的设计也考虑到了这种情况，为此专门在矩阵计算单元后面增加了一组累加器单元，可以实现将上一次的中间结果与当前的结果相累加，总共累加的次数可以由软件控制，并在累加完成之后将最终结果写入到输出缓冲区中。在卷积计算过程中，累加器可以完成加偏置的累加计算。

矩阵计算单元可以快速完成16\*16的矩阵相乘。但当超过16\*16大小的矩阵利用该单元进行计算时，则需要事先按照特定的数据格式进行矩阵的存储，并在计算的过程中以特定的分块方式进行数据的读取。如图1-8所示，矩阵A展示的切割和排序方式称作“大Z小Z”， 直观的看就是矩阵A的各个分块之间按照行的顺序排序，称之为“大Z”方式；而每个块的内部数据也是按照行的方式排列，称为“小Z”方式。与之形成对比的是矩阵B的各个分块之间按照行排序，而每个块的内部按照列排序，称为“大Z小N”的排序方式。按照矩阵计算的一般法则，如此排列的A、B矩阵相乘之后得到的结果矩阵C将会呈现出各个分块之间按照列排序，而每个块内部按照行排序的格式，称为“大N小Z”的排列方式。



存储格式要求

在利用矩阵计算单元进行大规模的矩阵运算时，由于矩阵计算单元的容量有限，往往不能一次存放下整个矩阵，所以也需要对矩阵进行分块并采用分步计算的方式，如图1-9所示， 将矩阵A和矩阵B都等分成同样大小的块，每一块都可以是一个16\*16的子矩阵，排不满的地方可以通过补零实现。首先求C1结果子矩阵，需要分两步计算：第一步将A1和B1搬移到矩阵计算单元中，并算出 A1\*B1 的中间结果；第二步将A2和B2搬移到矩阵计算单元中，再次计算 A2\*B2，并把计算结果累加到上一次 A1\*B1 的中间结果，这样才完成结果子矩阵C1的计算，之后将C1写入输出缓冲区。由于输出缓冲区容量也有限，所以需要尽快将C1子矩阵写入内存中，便于留出空间接受下一个结果子矩阵C2。同理依次类推可以完成整个大规模矩阵乘法的运算。



矩阵分块计算

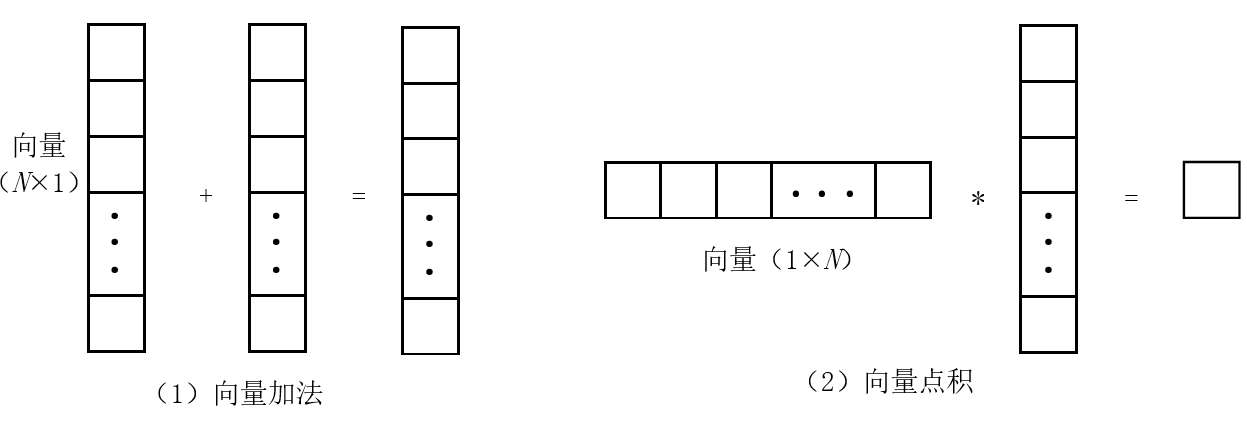
除了支持FP16类型的运算，矩阵计算单元也可以支持诸如INT8等更低精度类型的输入数据。对于INT8，矩阵计算单元可以一次完成一个16\*32矩阵与一个32\*16的矩阵相乘运算。程序员可以根据深度神经网络对于精度的要求来适当调整矩阵计算单元的运算精度，从而可以获得更加出色的性能。

矩阵计算单元除了支持FP16和INT8的运算之外，还同时支持UINT8、INT4 和U2数据类型计算。在U2数据类型下，只支持对两比特U2类型权重（Weight）的计算。由于现代轻量级神经网络权重为两比特的情况比较普遍，所以在计算中先将U2权重数据转换成FP16或者INT8后再进行计算。

#### 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。

如图 1-10 所示，向量计算单元可以快速完成两个FP16类型的向量运算。如图1-2 所示，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区中。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。向量计算单元支持的寻址模式包括了向量连续寻址和固定间隔寻址；在特殊情形下，对于地址不规律的向量，向量计算单元也 提供了向量地址寄存器寻址来实现向量的不规则寻址。



向量运算示例

如图1-2所示，向量计算单元可以作为矩阵计算单元和输出缓冲区之间的数据通路和桥梁。矩阵运算完成后的结果在向输出缓冲区传递的过程中，向量计算单元可以顺便完成在深度神经网络尤其是卷积神经网络计算中常用的ReLU激活函数、池化等功能并实现数据格式的转换。经过向量计算单元处理后的数据可以被写回到输出缓冲区或者矩阵计算单元中，以等待下一次运算。所有的这些操作都可以通过软件配合相应的向量单元指令来实现。向量计算单元提供了丰富的计算功能，也可以实现很多特殊的计算函数，从而和矩阵计算单元形成功能互补，全面完善了AI Core对非矩阵类型数据计算的能力。

#### 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。

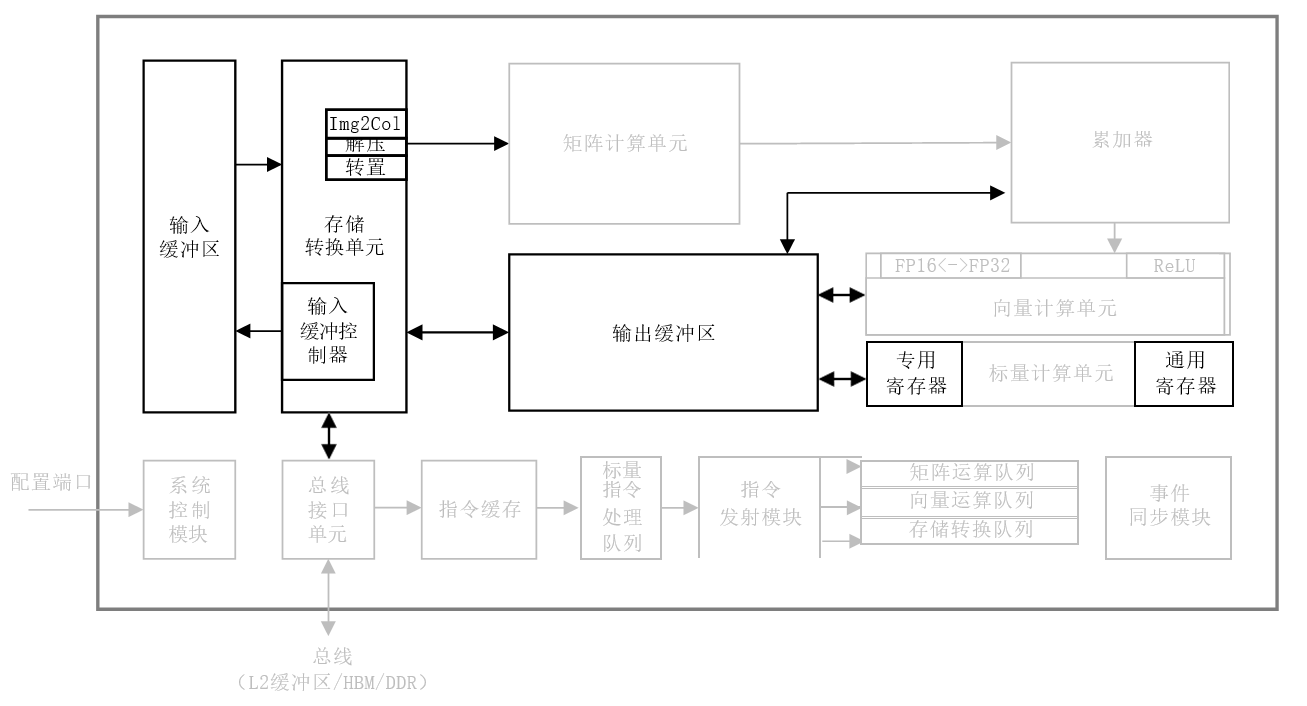
AI Core 中具有代表性的专用寄存器包括CoreID（用于标识不同的 AI Core），VA（向量地址寄存器）以及 STATUS（AI Core 运行状态寄存器）等。软件可以通过监视这些专用寄存器来控制和改变 AI Core 的运行状态和模式。

### 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。众所周知，几乎所有的深度学习算法都是数据密集型的应用。对于昇腾AI芯片来说，合理设计的数据存储和传输结构对于最终系统运行的性能至关重要。不合理的设计往往成为性能瓶颈，从而白白浪费了片上海量的计算资源。AI Core通过各种类型分布式缓冲区之间的相互配合，为深度神经网络计算提供了大容量和及时的数据供应，为整体计算性能消除了数据流传输的瓶颈，从而支撑了深度学习计算中所需要的大规模、高并发数据的快速有效提取和传输。

#### 存储单元

芯片中的计算资源要想发挥强劲算力，必要条件是保证输入数据能够及时准确的出现在计算单元里。达芬奇架构通过精心设计的存储单元为计算资源保证了数据的供应，相当于AI Core中的后勤系统。AI Core 中的存储单元由存储控制单元、缓冲区和寄存器组成，如图1-11中的加粗显示。存储控制单元通过总线接口可以直接访问AI Core之外的更低层级的缓存，并且也可以直通到DDR 或HBM 从而可以直接访问内存。存储控制单元中还设置了存储转换单元，其目的是将输入数据转换成AI Core中各类型计算单元所兼容的数据格式。缓冲区包括了用于暂存原始图像特征数据的输入缓冲区，以及处于中心的输出缓冲区来暂存各 种形式的中间数据和输出数据。AI Core中的各类寄存器资源主要是标量计算单元在使用。所有的缓冲区和寄存器的读写都可以通过底层软件显式的控制，有经验的程序员可以通过巧妙的编程方式来防止存储单元中出现读写冲突而影响流水线的进程。对于类似卷积和矩阵这样规律性强的计算模式，高度优化的程序可以实现全程无阻塞的流水线执行。



存储单元结构

图1-11中的总线接口单元作为AI Core 的“大门”，是一个与系统总线交互的窗口，并以此通向外部世界。AI Core通过总线接口从外部L2缓冲区、DDR或HBM中读取或者写回数据。总线接口在这个过程中可以将AI Core内部发出的读写请求转换为符合总线要求的外部读写请求，并完成协议的交互和转换等工作。

输入数据从总线接口读入后就会经由存储转换单元进行处理。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。存储转换单元还可以 控制AI Core内部的输入缓冲区，从而实现局部数据的缓存。

在深度神经网络计算中，由于输入图像特征数据通道众多且数据量庞大，往往会采用输入缓冲区来暂时保留需要频繁重复使用的数据，以达到节省功耗、提高性能的效果。当输入缓冲区被用来暂存使用率较高的数据时，就不需要每次通过总线接口到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。另外，当存储转换单元进行数据的格式转换操作时，会产生巨大的带宽需求，达芬奇架构要求源数据必须被存放于输入缓冲区中，才能够进行格式转换，而输入缓冲控制器负责控制数据流入输入缓冲区中。输入缓冲区的存在有利于将大量用于矩阵计算的数据一次性的被搬移AI Core内部， 同时利用固化的硬件极高的提升了数据格式转换的速度，避免了矩阵计算单元的阻塞，消除了由于数据转换过程缓慢而带来的性能瓶颈。

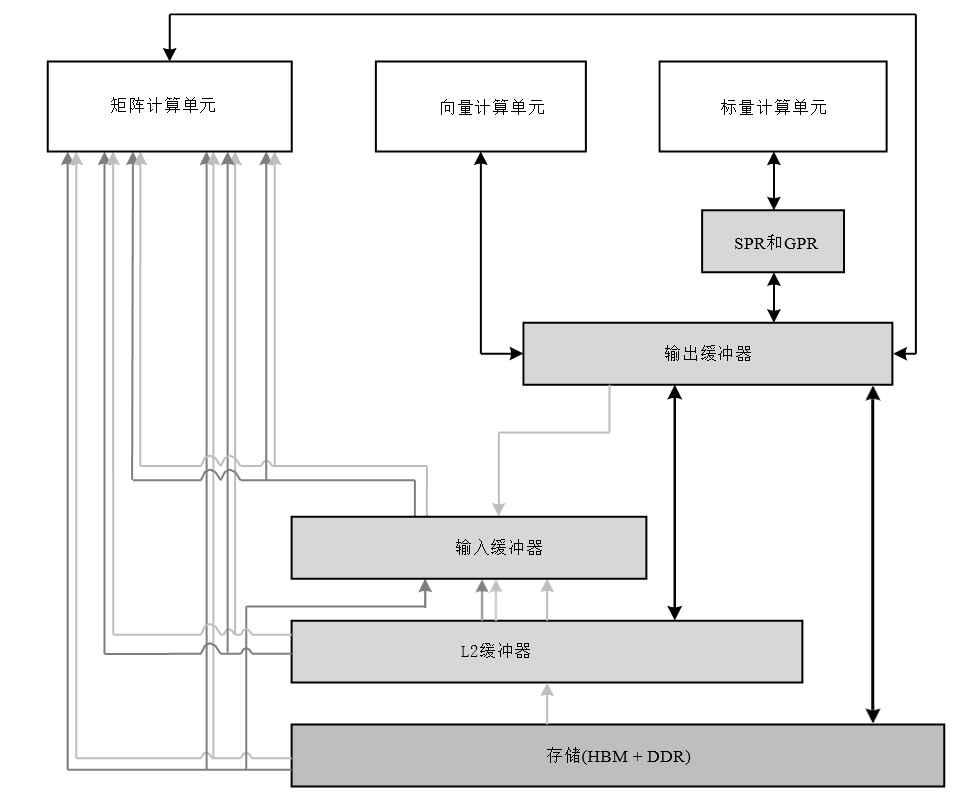
在神经网络中往往可以把每层计算的中间结果放在输出缓冲区中，从而在进入下一层计算时方便的获取数据。由于通过总线读取数据的带宽低，延迟大，通过充分利用输出缓冲区就可以大大提升计算效率。

在矩阵计算单元还包含有直接的供数寄存器，提供当前正在进行计算的大小为16\*16的左、右输入矩阵。在矩阵计算单元之后，累加器也含有结果寄存器，用于缓存当前计算的大小为16\*16 的结果矩阵。在累加器配合下可以不断的累积前次矩阵计算的结果，这在卷积神经网络的计算过程中极为常见。在软件的控制下，当累积的次数达到要求后，结果寄存器 中的结果可以被一次性的传输到输出缓冲区中。

AI Core 中的存储系统为计算单元提供源源不断的数据，高效适配计算单元的强大算力， 综合提升了 AI Core 的整体计算性能。与谷歌 TPU 设计中的统一缓冲区设计理念相类似， 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。

达芬奇架构通过存储转换单元中内置的定制电路，在进行数据传输的同时，就可以实现诸如 Img2Col 或者其它类型的格式转化操作，不光是节省了格式转换过程中的消耗，同时也节省了数据转换的指令开销。这种能将数据在传输的同时进行转换的指令称为随路指令。硬件单元对随路指令的支持为程序设计提供了便捷性。

#### 数据通路



基本数据通路图

数据通路指的是AI Core在完成一个计算任务时，数据在AI Core中的流通路径。前文已经以矩阵相乘为例简单介绍了数据的搬运路径。图1-12展示了达芬奇架构中一个AI Core内完整的数据传输路径。这其中包含了DDR或HBM，以及L2缓冲区，这些都属于AI Core核外的数据存储系统。图中其它各类型的数据缓冲区都属于核内存储系统。

核外存储系统中的数据可以通过LOAD指令被直接搬运到矩阵计算单元中进行计算， 输出的结果会被保存在输出缓冲区中。除了直接将数据通过LOAD指令发送到矩阵计算单元中，核外存储系统中的数据也可以通过LOAD指令先行传入输入缓冲区，再通过其它指令传输到矩阵计算单元中。这样做的好处是利用大容量的输入缓冲区来暂存需要被矩阵计算单元反复使用的数据。

矩阵计算单元和输出缓冲区之间是可以相互传输数据的。由于矩阵计算单元容量较小，部分矩阵运算结果可以写入输出缓冲区中，从而提供充裕的空间容纳后续的矩阵计算。当然也可以将输出缓冲区中的数据再次搬回矩阵计算单元作为后续计算的输入。输出缓冲区和向量计算单元、标量计算单元以及核外存储系统之间都有一条独立的双向数据通路。输出缓冲区中的数据可以通过专用寄存器或通用寄存器进出标量计算单元。

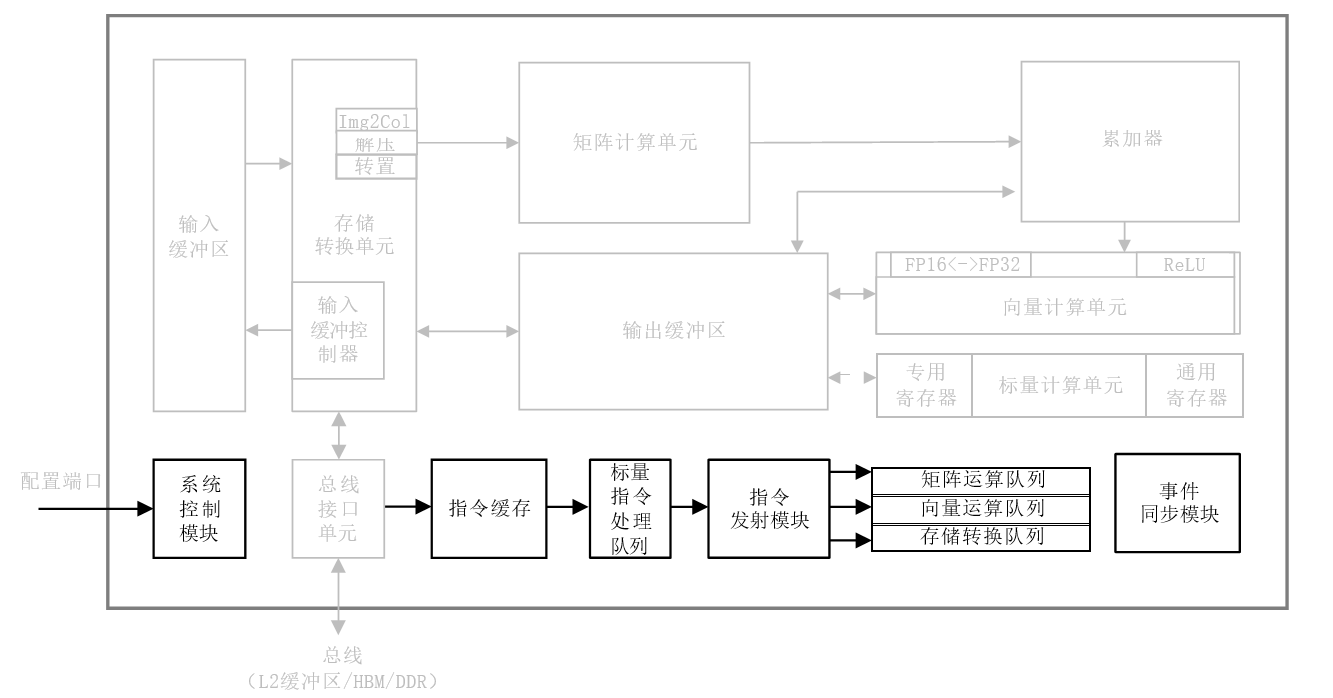
值得注意的是，AI Core中的所有数据如果需要向外部传输，都必须经过输出缓冲区，才能够被写回到核外存储系统中。例如输入缓冲区中的图像特征数据如果需要被输出到系统内存中，则需要先经过矩阵计算单元处理后存入输出缓冲区中，最终从输出缓冲区写回到核外存储系统中。在AI Core中并没有一条从输入缓冲区直接写入到输出缓冲区的数据通路。因此输出缓冲区作为AI Core数据流出的闸口，能够统一的控制和协调所有核内数据的输出。

达芬奇架构数据通路的特点是多进单出，数据流入AI Core可以通过多条数据通路，可以从外部直接流入矩阵计算单元、输入缓冲区和输出缓冲区中的任何一个，流入路径的方式比较灵活，在软件的控制下由不同数据流水线分别进行管理。而数据输出则必须通过输出缓冲区，最终才能输出到核外存储系统中。

这样设计的理由主要是考虑到了深度神经网络计算的特征。神经网络在计算过程中，往往输入的数据种类繁多并且数量巨大，比如多个通道、多个卷积核的权重和偏置值以及多个通道的特征值等，而 AI Core中对应这些数据的存储单元可以相对独立且固定，可以通过并行输入的方式来提高数据流入的效率，满足海量计算的需求。AI Core中设计多个输入数据通路的好处是对输入数据流的限制少，能够为计算源源不断的输送源数据。与此相反，深度神经网络计算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。根据神经网络输出数据的特点，在AI Core中设计了单输出的数据通路，一方面节约了芯片硬件资源，另一方面可以统一管理输出数据，将数据输出的控制硬件降到最低。

综上，达芬奇架构中的各个存储单元之间的数据通路以及多进单出的核内外数据交换机制是在深入研究了以卷积神经网络为代表的主流深度学习算法后开发出来的，目的是在保障数据良好的流动性前提下，减少芯片成本、提升计算性能、降低控制复杂度。

### 控制单元



控制单元逻辑图

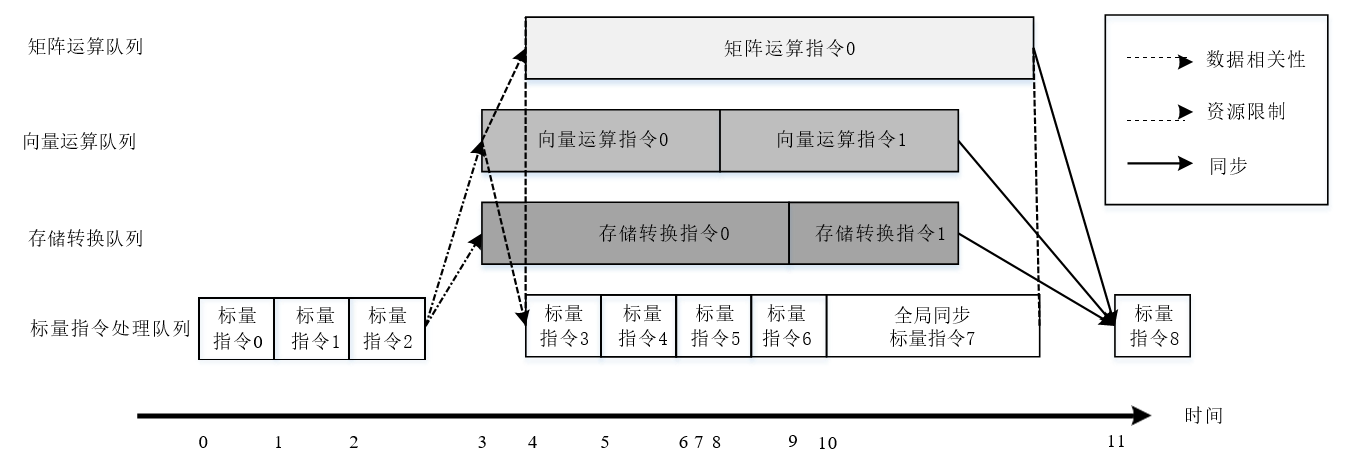
在达芬奇架构下，控制单元为整个计算过程提供了指令控制，相当于AI Core的司令部，负责整个AI Core的运行，起到了至关重要的作用。控制单元的主要组成部分为系统控制模块、指令缓存、标量指令处理队列、指令发射模块、矩阵运算队列、向量运算队列、存储转 换队列和事件同步模块，如图1-13中加粗所示。

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。多条指令从系统内存通过总线接口进入到AI Core的指令缓存中并等待后续硬件快速自动解码或运算。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。在进入指令发射模块之前，所有指令都作为普通标量指令被逐条顺次处理。标量队列将这些指令的地址和参数解码配置好后，由指令发射模块根据指令的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，如图1-13所示。

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。矩阵计算指令进入矩阵运算队列，向量计算指令进入向量运算队，存储转换指令进入存储转换队列，同一个指令执行队列中的指令是按照进入队列的顺序进行执行的，不同指令执行队列之间可以并行执行，通过多个指令执行队列的并行执行可以提升整体执行效率。

当指令执行队列中的指令到达队列头部时就进入真正的指令执行环节，并被分发到相应的执行单元中，如矩阵计算指令会发送到矩阵计算单元，存储转换指令会发送到存储转换单元。不同的执行单元可以并行的按照指令来进行计算或处理数据，同一个指令队列中指令执 行的流程被称作为指令流水线。

对于指令流水线之间可能出现的数据依赖，达芬奇架构的解决方案是通过设置事件同步模块来统一协调各个流水线的进程。事件同步模块时刻控制每条流水线的执行状态，并分析不同流水线的依赖关系，从而解决数据依赖和同步的问题。比如矩阵运算队列的当前指令需要依赖向量计算单元的结果，在执行过程中，事件同步控制模块会暂停矩阵运算队列执行流程，要求其等待向量计算单元的结果。而当向量计算单元完成计算并输出结果后，此时事件同步模块则通知矩阵运算队列需要的数据已经准备好，可以继续执行。在事件同步模块准许放行之后矩阵运算队列才会发射当前指令。在达芬奇架构中，无论是流水线内部的同步还是流水线之间的同步，都是通过事件同步模块利用软件控制来实现的。



指令执行与控制

如图1-14所示，示意了四条流水线的执行流程。标量指令处理队列首先执行标量指令0、1和2三条标量指令，由于向量运算队列中的指令0和存储转换队列中的指令0与标量指令2存在数据依赖性，需要等到标量指令2完成才能发射并启动。由于指令是被顺序发射的，因此只能等到时刻4时才能发射并启动矩阵运算指令0和标量指令3，这时四条指令流水线可以并行执行。直到标量指令处理队列中的全局同步标量指令7生效后，由事件同步模块对矩阵流水线、向量流水线和存储转换流水线进行全局同步控制，需要等待矩阵运算指令0、向量运算指令1和存储转换指令1都执行完成后，事件同步模块才会允许标量流水线继续执行标量指令8。

在控制单元中还存在一个系统控制模块。在AI Core运行之前，需要外部的任务调度器来控制和初始化AI Core的各种配置接口，如指令信息、参数信息以及任务块信息等。这里的任务块是指AI Core中的最小的计算任务粒度。在配置完成后，系统控制模块会控制任务块的执行进程，同时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。如果在执行过程中出现了错误，系统控制模块将会把执行的错误状态报告给任务调度器，进而反馈当AI Core的状态信息给整个昇腾AI芯片系统。

# 昇腾软件架构

昇腾AI芯片的达芬奇架构在硬件设计上采用了计算资源的定制化设计，功能执行与硬件高度适配，为卷积神经网络计算性能的提升提供了强大的硬件基础。对于一个神经网络的算法，从各种开源框架，到神经网络模型的实现，再到实际芯片上的运行，中间需要多层次 的软件结构来管理网络模型、计算流以及数据流，神经网络软件流为从神经网络到昇腾AI芯片的落地实现过程提供了有力支撑，同时开发工具链为基于昇腾 AI 芯片的神经网络应用开发带了诸多便利，而神经网络软件流和开发工具链构成了昇腾 AI 芯片的基础软件栈，从上而下支撑起整个芯片的执行流程。

## 昇腾AI软件栈总览

为了使昇腾AI芯片发挥出极佳的性能，设计一套完善的软件解决方案是非常重要的。一个完整的软件栈包含计算资源、性能调优的运行框架以及功能多样的配套工具。昇腾AI芯片的软件栈可以分为神经网络软件流、工具链以及其它软件模块。

神经网络软件流主要包含了流程编排器（Matrix），框架管理器（Framework），运行管理器（Runtime）、数字视觉预处理模块（Digital Vision Pre-Processing，DVPP）、张量加速引擎（Tensor Boost Engine，TBE）以及任务调度器（Task Scheduler，TS）等功能模块。神经网络软件流主要用来完成神经网络模型的生成、加载和执行等功能。工具链主要为神经网络实现过程提供了辅助便利。

如图2-1所示，这些主要组成部分在软件栈中功能和作用相互依赖，承载着数据流、计算流和控制流。



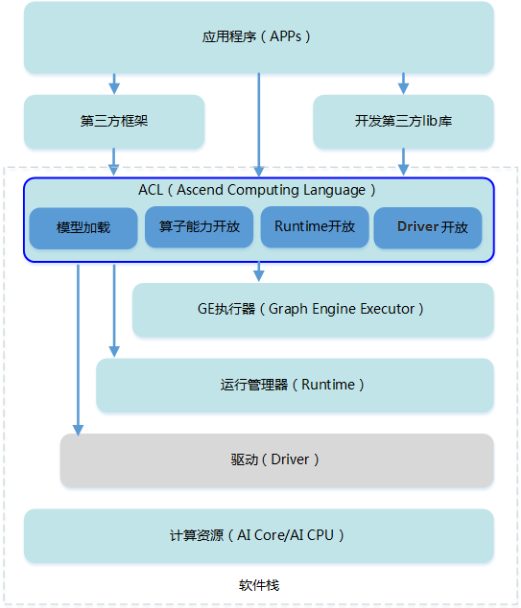
昇腾AI处理器软件逻辑架构

计算资源层主要实现系统对数据的处理和对数据的运算执行。计算设备主要包括AI Core：执行NN类算子；AI CPU：执行CPU算子；DVPP：视频/图像编解码、预处理。通信链路主要包括PCIe：芯片间或芯片与CPU间高速互联；HCCS：实现芯片间缓存一致性功能；RoCE：实现芯片内存 RDMA功能。

芯片使能层实现解决方案对外能力开放，以及基于计算图的业务流的控制和运行。AscendCL昇腾计算语言库是开放编程框架，提供Device/Context/Stream/内存等的管理、模型及算子的加载与执行、媒体数据处理、Graph管理等API库，供用户开发深度神经网络应用；图优化和编译统一的IR接口对接不同前端，支持TensorFlow/ Caffe/MindSpore表达的计算图的解析/优化/编译，提供对后端计算引擎最优化部署能力；数字视觉预处理实现视频编解码(VENC/VDEC)、JPEG编解码(JPEGD/E)、PNG解码(PNGD)、VPC(预处理)；执行引擎包括两个部分分别是运行管理器和任务调度器，Runtime为神经网络的任务分配提供资源管理通道，Task Scheduler主要计算Task序列的管理和调度以及执行。

应用层包括基于Ascend平台开发的各种应用，以及Ascend提供给用户进行算法开发、调优的应用类工具。推理应用是基于AscendCL提供的API构建推理应用；AI框架包括Tensorflow、Caffe、Mindspore以及第三方框架；模型小型化工具实现对模型进行量化，加速模型；AutoML工具是基于MindSpore自动学习工具，根据昇腾芯片特点进行搜索生成亲和性网络，充分发挥昇腾性能；加速库是基于AscendCL构建的加速库（当前支持Blas加速库）；MindStudio提供给开发者的集成开发环境和调试工具，可以通过MindStudio进行离线模型转换、离线推理算法应用开发调试、算法调试、自定义算子开发和调试、日志查看、性能调优、系统故障查看等

## ACL子系统



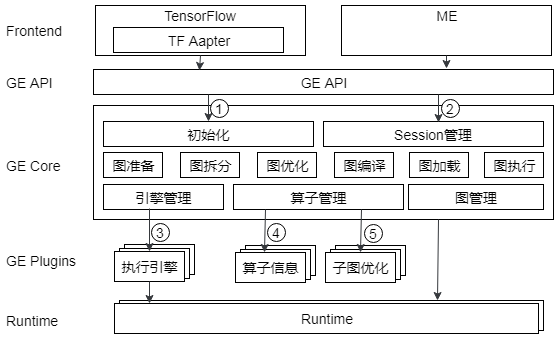
逻辑架构图

其中API适配管理：ACL是api或算子开放的入口。ACL提供的是分层开放能力的管控，通过不同的组件对不同的使能部件进行对接。包含GE能力开放、算子能力开放、Runtime能力开放、driver能力开放等。GE能力开放：处理基于图及session的开放，能力引擎在GE侧，但接口的开放是通过ACL，包含图编辑、图编译、图执行的能力。算子能力开放：算子能力实现在CANN中，但算子能力开放是通过ACL。Runtime能力开放：处理基于stream的设备能力、内存、event等资源能力开发诉求，对app屏蔽底层实现。Driver能力开放：使用户程序能够使用驱动提供的队列等管理机制，屏蔽硬件架构的复杂性和异构性。

ACL（Ascend Computing Language）提供Device管理、Context管理、Stream管理、内存管理、模型加载与执行、算子加载与执行、媒体数据处理等C++ API库供用户开发深度神经网络应用，通过加载模型推理实现目标识别、图像分类等功能。用户可以通过第三方框架调用ACL接口，以便使用昇腾AI处理器的计算能力；用户还可以使用ACL封装实现第三方lib库，以便提供昇腾AI处理器的运行管理、资源管理能力。在运行应用时，ACL调用GE执行器提供的接口实现模型和算子的加载与执行、调用运行管理器的接口实现Device管理/Context管理/Stream管理/内存管理等。

ACL提供的是分层开放能力的管控，通过不同的组件对不同的使能部件进行对接。包含GE能力开放、算子能力开放、Runtime能力开放、Driver能力开放等。

## GE子系统



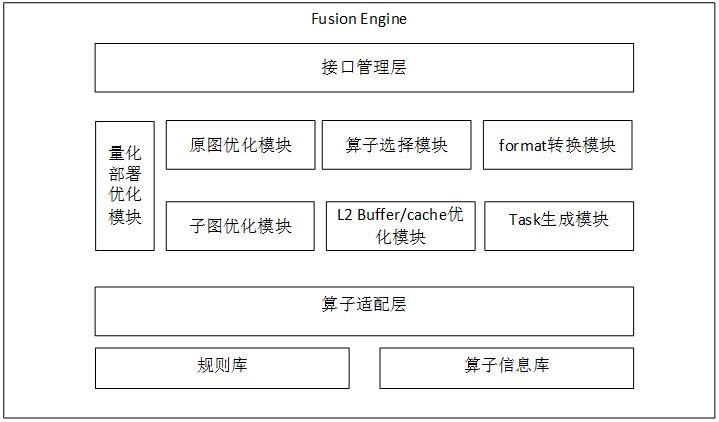
MindSpore

GE子系统

GE（Graph Engine）作为图编译和运行的控制中心，提供运行环境管理、执行引擎管理、算子库管理、子图优化管理、图操作管理和图执行控制。GE通过统一的接口提供多前端的支持，不同的前端框架可以通过适配层完成不同格式图到GE IR Graph的转换。GE API对外呈现GE Core中初始化、Session管理模块的接口，支持运行环境初始化，Session创建、销毁，图添加执行。GE Core从上往下，共分为三层，分别是执行控制层、业务功能层、数据管理层。

执行控制层，提供API接口实现逻辑的控制，通过Runtime、业务功能模块、数据管理模块的接口，完成功能的实现。业务功能层，为图执行提供最优执行引擎匹配，端到端执行路径优化，提供最低执行开销，支持不同的物理运行环境部署。数据管理层，包含对外部插件的管理（执行引擎、算子库）及GE执行过程中需要的内部数据管理（图），对业务功能提供支持。

## FE子系统



FE子系统

FE（Fusion Engine）：为了业务的编排，抽象出执行引擎FE。它是一个逻辑概念，里面分三个部件：控制引擎、计算引擎、IO引擎。每个部件中都包含有算子库。FE的功能框架，从业务层次上划分，可以分为三层：上层是接口层，对外提供各种接口；中间是业务层，主要是两块功能：图优化和Task生成；底层是数据层，维护着计算引擎信息、融合规则信息和算子信息库。

FE定位于AI Core的数据引擎，它提供图的优化分析、管理算子融合规则、算子融合功能、算子信息库管理、使能自定义算子等功能，提供如下功能：

接口管理层：对GE提供了算子管理，优化管理，task生成等接口；

量化部署优化模块：提供INT8量化的部署优化，辅助量化工具完成对图的INT8量化处理；

原图优化模块：提供对整图的图融合处理；

算子选择模块：提供了对图上各节点优先选择aicore算子实现；

Format转换模块：提供了选择D芯片下算子实现支持的format和插入转换op的功能；

子图优化模块： 对GE拆分的aicore引擎子图进行优化处理；

L2 buffer/cache优化模块：对图上算子进行L2 buffer/cache的优化处理；

Task生成模块：提供了生成taskinfo的功能；

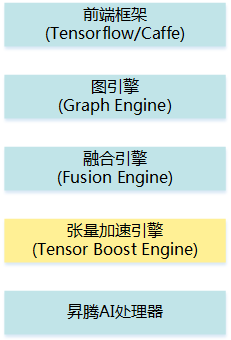
算子适配层：提供了适配TBE算子，自定义算子的功能；

规则库：提供了融合的规则列表；

算子信息库：提供了算子实现的信息库文件列表。

## TBE子系统

TBE (Tensor Boost Engine)：提供了基于TVM框架的自定义算子开发能力，通过TBE提供的API和自定义算子编程开发界面可以完成相应神经网络算子的开发。

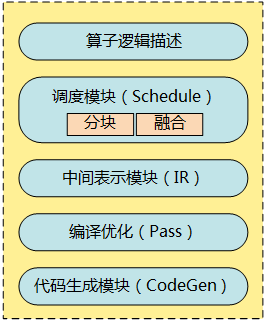


TBE在昇腾软件中的逻辑架构

其中前端框架包含第三方开源框架Tensorflow、Caffe。图引擎（Graph Engine：GE)是基于昇腾AI处理器软件栈对不同的机器学习框架提供统一的IR接口，对接上层网络模型框架，例如Tensorflow、Caffe。主要功能包括图准备、图拆分、图优化、图编译、图加载、图执行和图管理等（此处图指网络模型拓扑图）。

融合引擎（Fusion Engine：FE）负责对接GE和TBE算子，具备算子信息库的加载与管理、融合规则管理、原图融合和子图优化的能力。GE在子图优化阶段将子图传递给FE，FE根据算子信息库以及FE融合优化进行预编译，例如修改数据类型、插入转换算子等，该子图将再次传递给GE进行子图合并及子图优化。

张量加速引擎（TBE）通过IR定义为GE的图推导提供必要的算子信息，通过算子信息库和融合规则为FE提供子图优化信息和TBE算子调用信息，TBE生成的二进制对接昇腾AI处理器，最终生成网络在昇腾AI处理器上的执行任务。

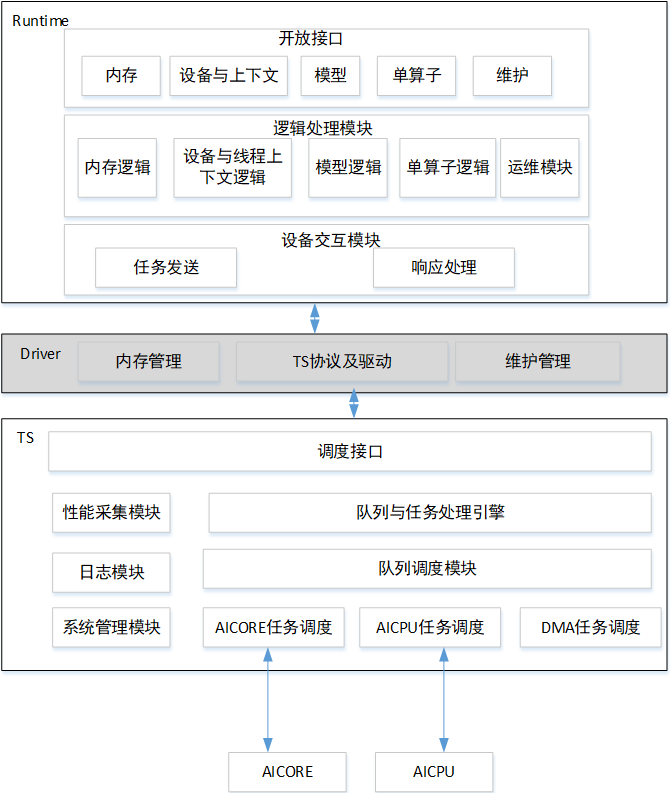


TBE功能框架

DSL模块是面向开发者，提供算子逻辑的编写的接口（Compute接口），使用接口来编写算子。

Schedule模块：用于描述指定shape下算子如何在昇腾AI处理器上进行切分，包括Cube类算子的切分、Vector类算子的切分，它们仍然使用的是社区提供的调度原语来描述。IR模块借用社区的IR来表示的，包括IR变形、AST树的维护等功能。编译优化（Pass）对生成的IR进行编译优化，优化的方式有双缓冲（Double Buffer）、流水线（Pipeline）同步、内存分配管理、指令映射、分块适配矩阵计算单元等。代码生成模块（CodeGen）CodeGen生成类C代码的临时文件，这个临时代码文件可以通过编译器生成算子的实现文件，可被网络模型直接加载调用。

## Runtime&TS子系统

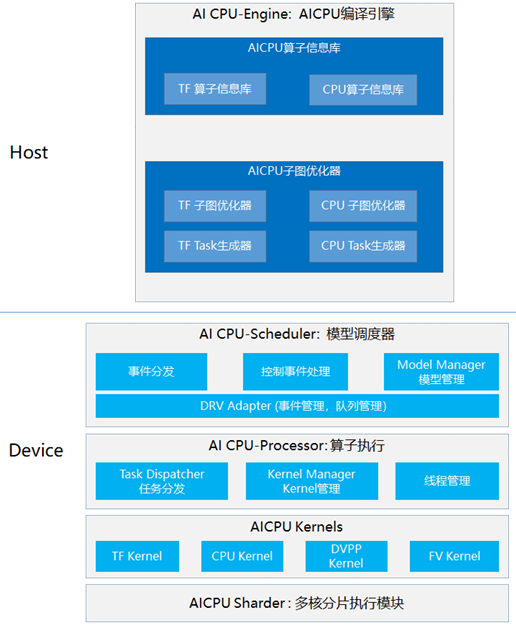


Runtime&TS子系统内

Runtime为神经网络的任务分配提供了资源管理通道。Runtime运行在应用程序的进程空间中，为应用程序提供了存储(Memory)管理、设备(Device)管理、执行流(Stream)管理、事件(Event)管理、核(Kernel)函数执行等功能。

Task Schedule运行在Device侧的任务调度CPU 上，负责将Runtime分发的具体任务进一步分发到AICPU 上。它也可以通过硬件任务调度器（HWTS）把任务分配到AI Core上执行，并在执行完成后返回任务执行的结果给运行管理器。通常Task Schedule处理的主要事务有AI Core任务、AI CPU 任务、内存复制任务、事件记录任务、事件等待任务、清理维护(Maintenance)任务和性能分析(Profiling)任务。

## AICPU子系统



AICPU子系统内

AICPU子系统提供两大功能：AICPU算子编译（部署在Host）和ACIPU调度执行（部署在Device）。

AICPU算子编译有AICPU算子信息库和AICPU图优化器组成。AICPU算子信息库包括TF算子信息库支持的TF算子名称、支持的format等。AICPU图优化器实现TF子图优化器，将TF子图优化成单function执行，减少task中断次数，提升AICPU算子执行效率。图优化器同时包含GE IR to TF算子的配置信息，可灵活的设置GE IR 与TF 算子的映射关系。

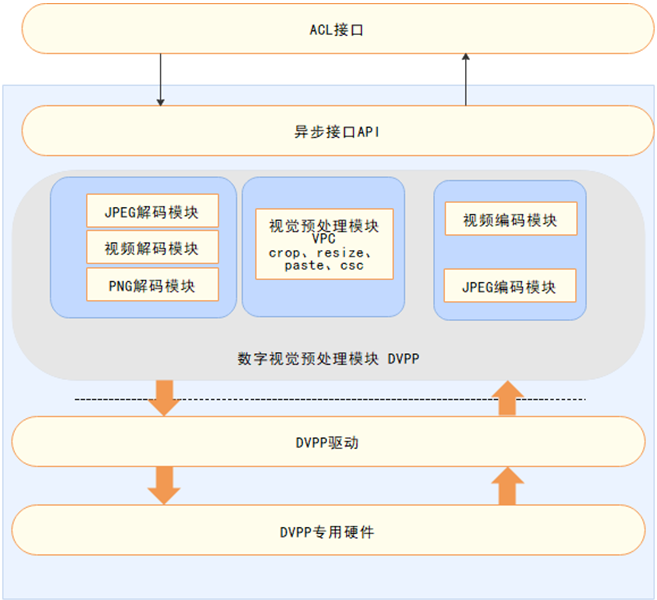
ACIPU调度执行（部署在Device）包括AICPU-Scheduler、AICPU-Processor和AICPU-Kernels。AICPU-Scheduler负责模型、stream等信息管理和模型状态管理；负责与TS交互控制命令，调度模型执行和结束。AICPU-Processor提供AICPU任务工作线程管理和Kernel注册管理。提供task分发功能，将CPU的task分发调度到各AICPU工作线程进行执行。AICPU-Kernels提供CPU算子，TF算子，DVPP算子，FV（短特征）算子的执行功能。

关于Host/Device是指在异构计算架构中, 昇腾AI处理器与CPU通过PCIe总线连接在一起来协同工作。Host是CPU所在位置称为主机端（Host），是指与昇腾AI处理器所在硬件设备相连的X86服务器、ARM服务器或者WindowsPC，利用昇腾AI处理器提供的NN(Neural-Network)计算能力完成业务。Device是指安装了昇腾AI处理器的设备，利用PCIe接口与Host连接，为Host提供NN计算能力。

## DVPP子系统

DVPP（Digital Vision Pre-Processing）主要实现视频解码（VDEC）、视频编码（VENC）、JPEG编解码（JPEGD/E）、PNG解码（PNGD）、VPC（预处理）。

DVPP对外提供的接口有视频解码（VDEC）支持H264、H265两种视频格式的解码；视频编码（VENC）实现YUV/YVU420图片数据的编码，支持H264、H265两种视频格式的编码；JPEG编码（JPEGE）将YUV格式图片编码成.jpg图片，支持YUV422 Packed、YUV420SP（NV12,NV21）。JPEG解码（JPEGD）实现.jpg、.jpeg、.JPG、.JPEG图片的解码，对于硬件不支持的格式，会使用软件解码。PNG解码（PNGD）实现PNG格式图片的硬件解码。支持RGBA、RGB格式的图片解码成RGBA、RGB格式。VPC(Vision Preprocessing Core)支持的功能有：抠图、缩放、叠加、黏贴、格式转换功能。



DVPP子系统