

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 处理器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。

3.1 昇腾 AI 处理器总览

昇腾 AI 处理器本质上是一个片上系统,如图 3-1 所示,主要可以应用在与图像、视频、语音、文字处理相关的场景。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。该处理器大致可以划分为: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。处理器可以采用 LPDDR4 高速主存控制器接口,价格较低。目前主流片上系统处理器的主存一般由 DDR(Double Data Rate, 双倍速率内存)或 HBM(High Bandwidth Memory, 高带宽存储器)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。其他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。

当该处理器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他单元实现数据互换。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 处理器集成了多个 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处理

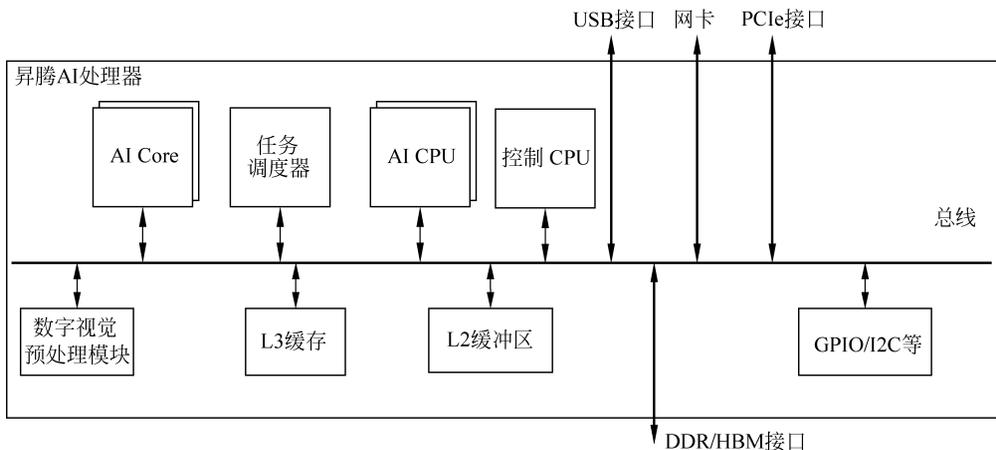


图 3-1 昇腾 AI 处理器逻辑图

器整体运行的控制 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,该处理器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需的常用计算,如矩阵相乘等。目前该处理器能对整型数(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,可以很方便地通过叠加模块的方法提高后续芯片的计算力。

针对神经网络参数量大、中间值多的特点,该处理器还特意为 AI 计算引擎配备了容量为 8MB 的片上缓冲区(L2 缓冲区),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,该处理器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 4K^① 分辨率的视频处理,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 处理器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

① 4K 指分辨率为 4096×2160 像素,表示超高清分辨率视频。

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。

3.2 达芬奇架构

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片 ASIC, 达芬奇架构本质上是为了适应某个特定领域中的常见应用和算法, 通常称为“特定域架构 (Domain Specific Architecture, DSA)”芯片。

昇腾 AI 处理器的计算核心主要由 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 3-2 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 和 FP16 的计算; 向量计算单元目前可以支持 FP16 和 FP32 以及多种整型数的计算。

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地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地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

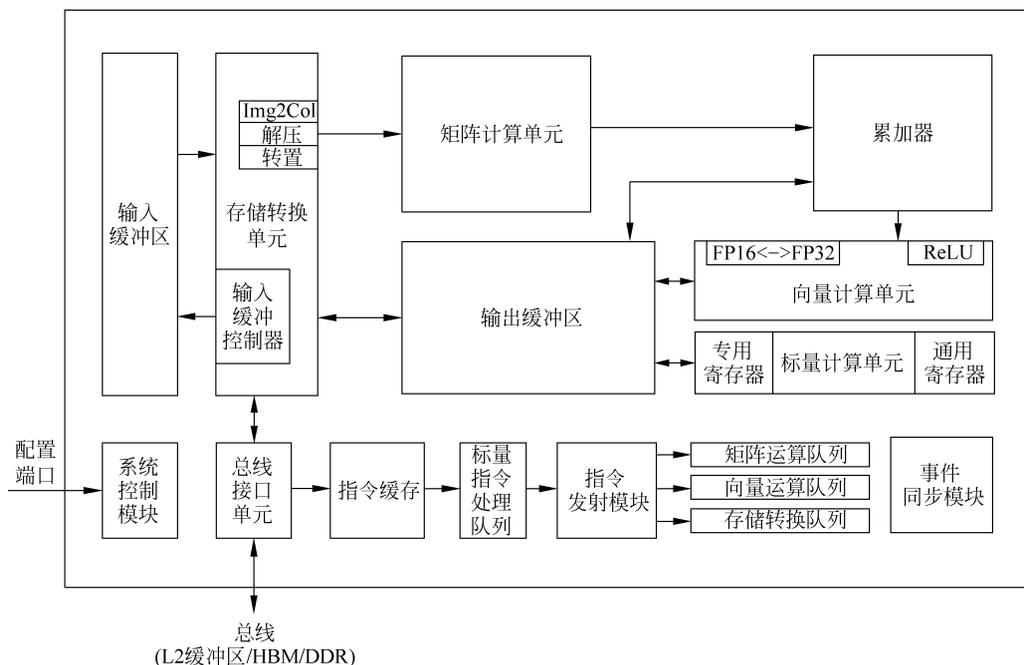


图 3-2 AI Core 架构图

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。比如前面提到 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 中是通过软件来实现的, 效率比较低。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神经网络中出现得极为频繁, 这种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, 指令将会分别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被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。

3.2.1 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 3-3 中的加粗部分所示。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别的标量数据运算和程序的流程控制。

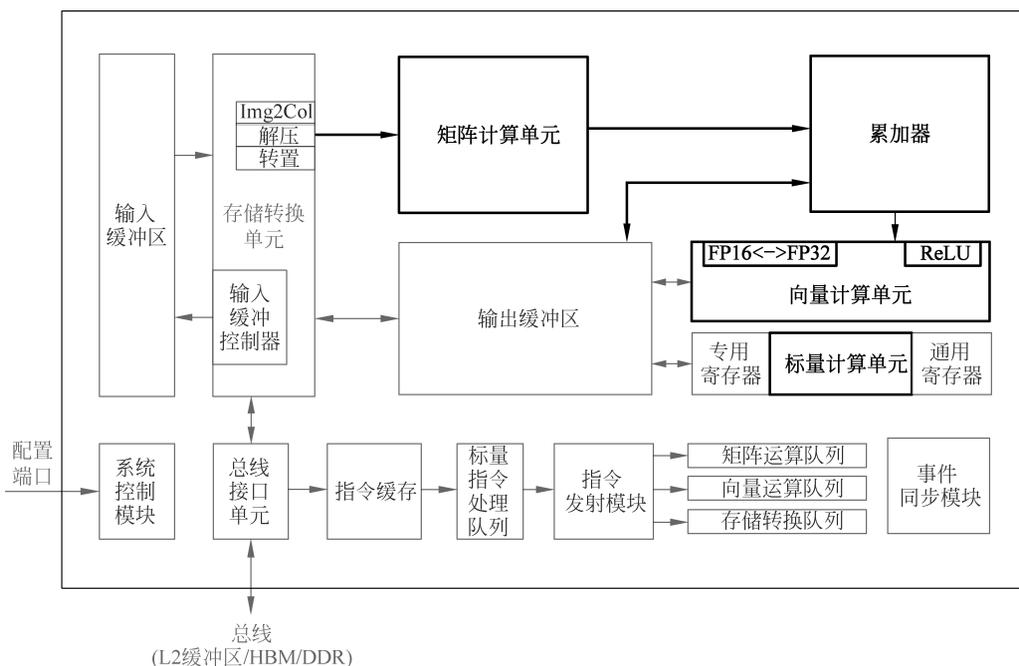


图 3-3 计算单元

1. 矩阵计算单元

1) 矩阵乘法

由于常见的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 3-4 表示一个矩阵 A 和另一个矩阵 B 之间的乘法运算 $C=A \times B$,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。

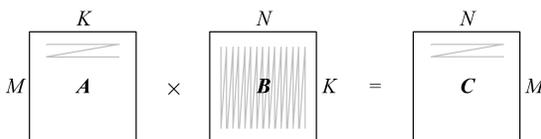


图 3-4 矩阵乘法示意图

在传统 CPU 中计算矩阵乘法的典型代码如代码 3-1 所示。

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

代码 3-1 CPU 计算矩阵乘法

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 $M \times K \times N$ 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时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 3-5 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。

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 3-6 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 B_0 、 B_1 、 B_2 和 B_3 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,尽可能地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,完成整个矩阵计算的全过程,最终得到结果矩阵 C 。矩阵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,可以高效地实现大规模的矩阵乘法计算,分块是一种常见的优化手段。

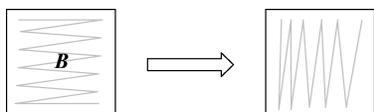
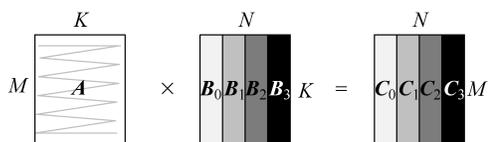
图 3-5 矩阵 B 存储方式

图 3-6 矩阵分块计算

2) 矩阵计算单元的计算方式

在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,这个延时是传统 GPU 无法避免的瓶颈。而异腾 AI 处理器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 处理器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 处理器的核心计算模块,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 $C = A \times$

B 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,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,可以由一条指令并行完成矩阵 **C** 的 256 个元素计算。

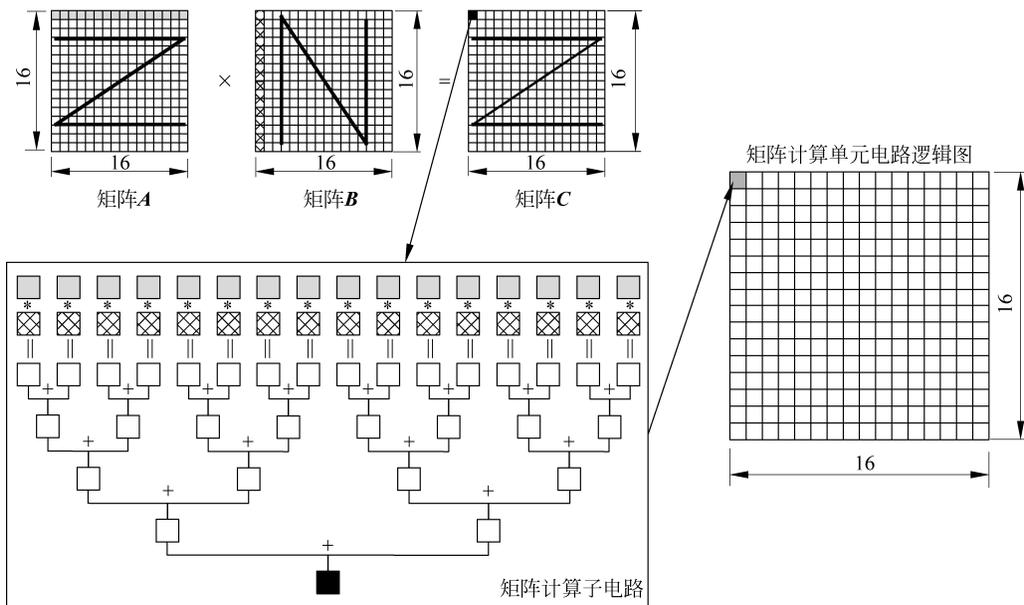


图 3-7 矩阵计算单元计算示意图

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 $C = A \times B + C$ 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输出缓冲区。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 3-8 所示,矩阵 **A** 展示的切割和排序方式称作“大 Z 小 Z”,直观地看就是矩阵 **A** 的各个分块之间按照行的顺序排序,称为“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,昇腾 AI 处理器内部专用电路可以实现将如此排列的 **A**、**B** 矩阵相乘之后得到结果矩阵 **C**,而矩阵 **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。

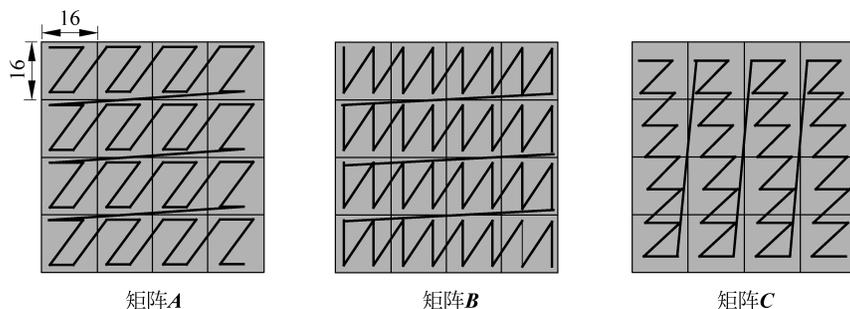


图 3-8 存储格式要求

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 3-9 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 C_1 结果子矩阵,需要分两步计算:第一步将 A_1 和 B_1 搬移到矩阵计算单元中,并算出 $A_1 \times B_1$ 的中间结果;第二步将 A_2 和 B_2 搬移到矩阵计算单元中,再次计算 $A_2 \times B_2$,并把计算结果累加到上一次 $A_1 \times B_1$ 的中间结果,这样才完成结果子矩阵 C_1 的计算,之后将 C_1 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 C_1 子矩阵写入内存中,便于留出空间接收下一个结果子矩阵 C_2 。同理,以此类推可以完成整个大规模矩阵乘法的运算。上述计算方式是最基本的分块矩阵计算方式,没有复用数据。在昇腾 AI 处理器的实际计算过程中,会在一定程度上复用矩阵 **A** 和矩阵 **B** 的分块矩阵。

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 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

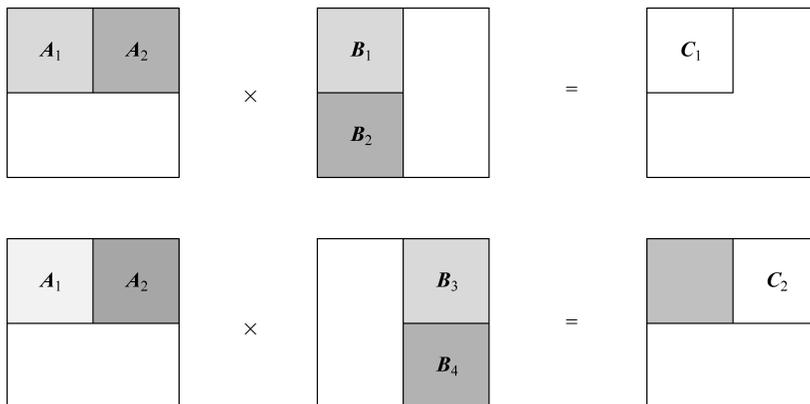


图 3-9 矩阵分块计算

FP16 或者 INT8 后再进行计算。

2. 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 3-10 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 3-2 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,对于地址不规则的向量,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。

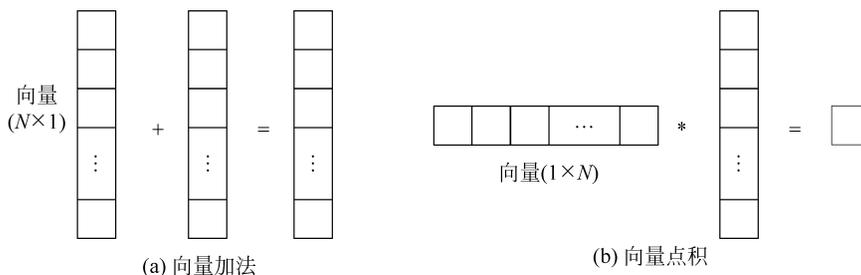


图 3-10 向量运算示例

如图 3-2 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。

3. 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 中其他功能性单元的执行流程。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。其他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 and 改变 AI Core 的运行状态和模式。

3.2.2 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 处理器来说,合理设计数据存储和传输结构对于系统的最终运行性能至关重要。不合理的设计往往成为系统性能瓶颈,从而白白浪费片上海量的计算资源。AI Core 通过各种类型的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所需要的大规模、高并发数据的快速有效提取和传输。

1. 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地出现在计算单元里。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 3-11 中的加粗部分所示。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。

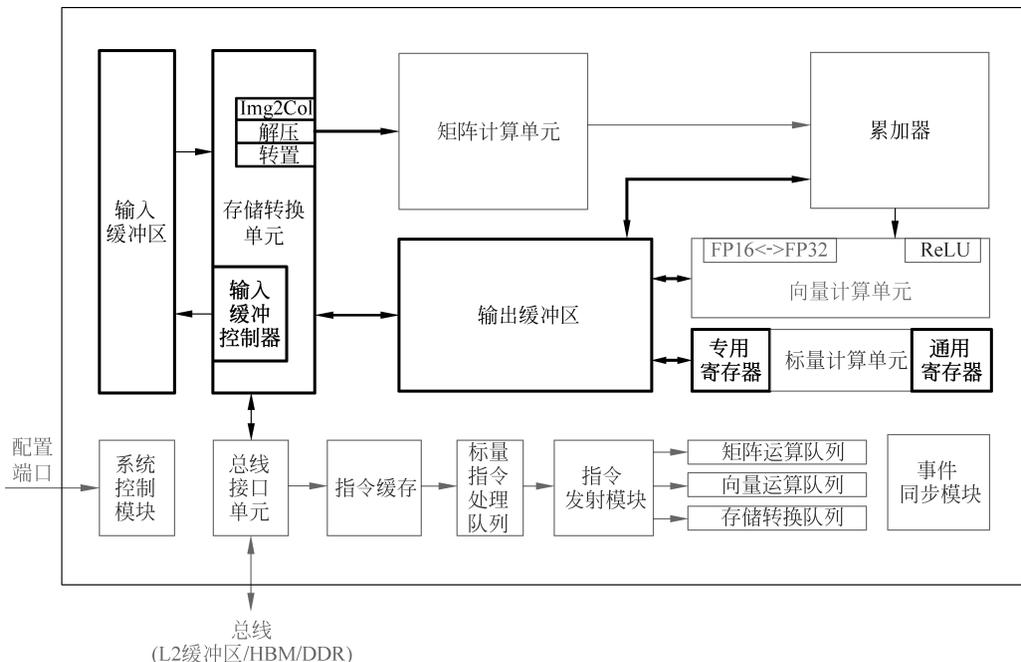


图 3-11 存储单元结构

所有的缓冲区和寄存器的读写都可以通过底层软件显式地控制,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。

图 3-11 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地搬移到 AI Core 内部，同时利用固化的硬件大幅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地获取数据。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提供数据的寄存器，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地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地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 或者其他类型的格式转化操作,不仅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。

2. 数据通路

数据通路指的是 AI Core 在完成一个计算任务时,数据在 AI Core 中的流电路径。前文已经以矩阵相乘为例简单介绍了数据的搬运路径。图 3-12 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 及 L2 缓冲区,这些都属于 AI Core 核外的数据存储系统。图中其他各类型的数据缓冲区都属于核内存储系统。

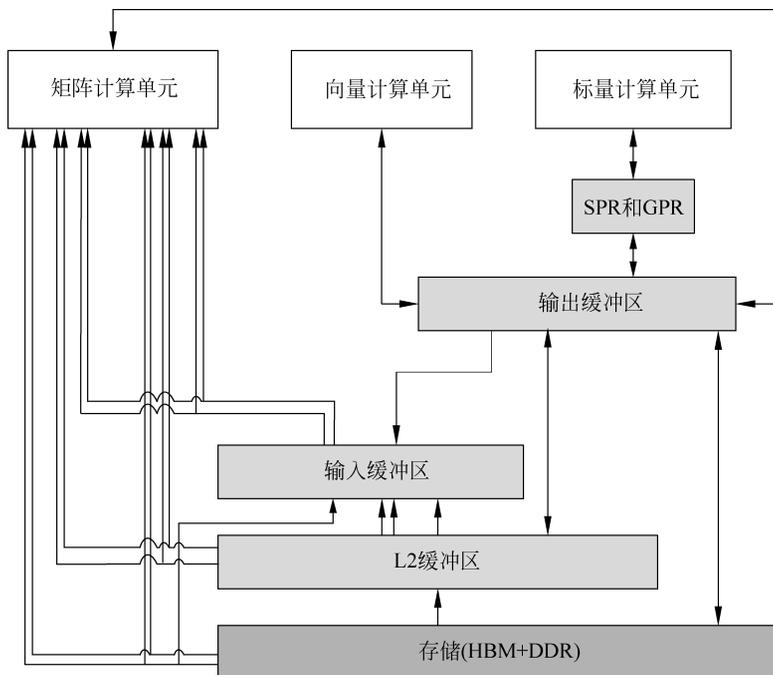


图 3-12 基本数据通路图

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,再通过其他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,能够统一地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地输送源数据。与此相反,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。

综上所述,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。

3.2.3 控制单元

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,控制单元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 3-13 中加粗部分所示。

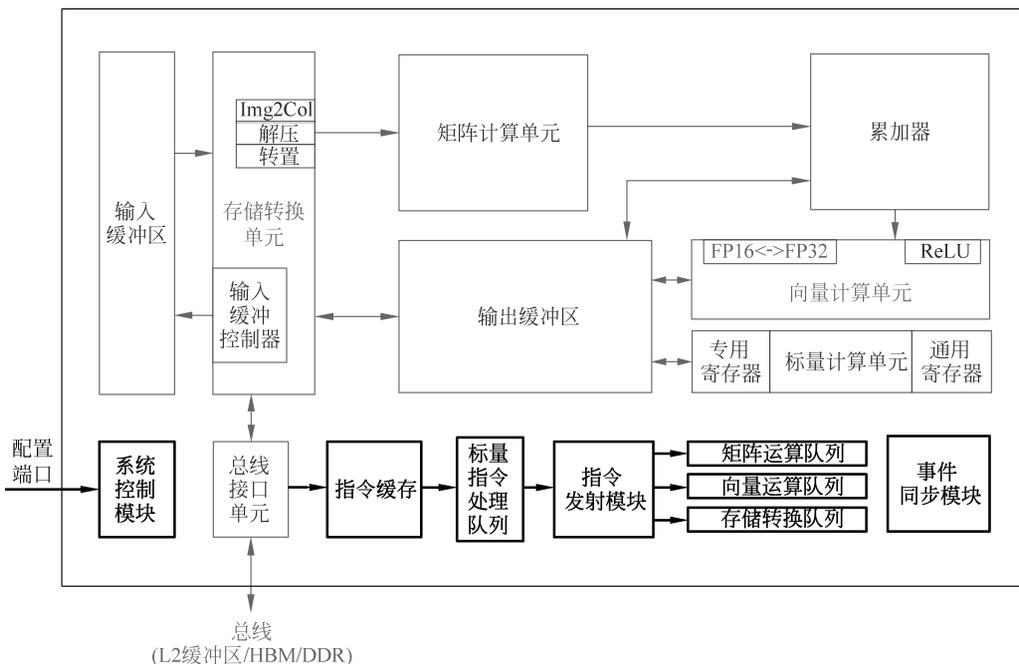


图 3-13 控制单元逻辑图

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。

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地按照指令来计算或处理数据,同一个指令队列中指令执行的流程被称之为指令流水线。

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。

如图 3-14 所示,示意了 4 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(时刻 3)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 4 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 and 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,进而反馈当前 AI Core 的状态信息给整个昇腾 AI 处理器系统。

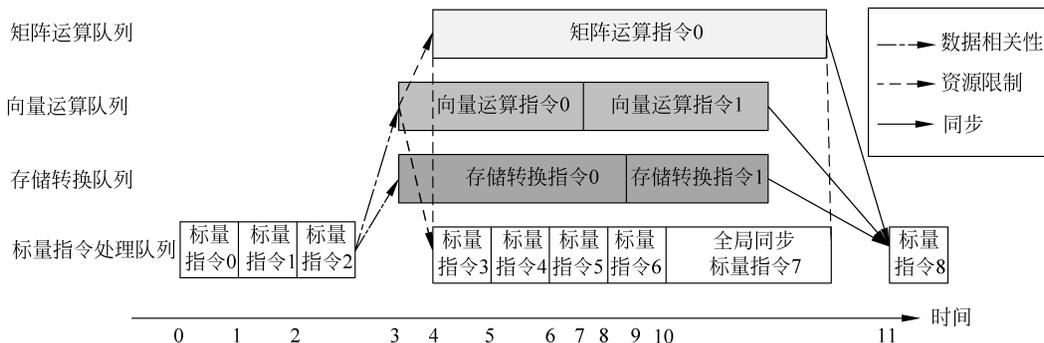


图 3-14 指令执行与控制

3.2.4 指令集设计

任何程序在处理器芯片中执行计算任务时,都需要通过特定的规范转化成硬件能够理解并处理的语言,这种语言就称为指令集架构(Instruction Set Architecture, ISA),简称指令集。指令集包含了数据类型、基本操作、寄存器、寻址模式、数据读写方式、中断、异常处理以及外部 IO 等,每条指令都会描述处理器的一种特定功能。指令集是计算机程序能够调用的处理器全部功能的集合,是处理器功能的抽象模型,也是作为计算机软件与硬件的接口。

指令集可以分为精简指令集(Reduced Instruction Set Computer, RISC)和复杂指令集(Complex Instruction Set Computer, CISC)。精简指令集的特点是单指令功能简单,执行速度快,编译效率高,不能直接操作内存,仅能通过指令来访问内存。常见的精简指令集有 ARM、MIPS、OpenRISC 以及 RISC-V 等。复杂指令集的特点是单指令功能强大且复杂,指令执行周期长,可以直接操作内存,常见的复杂指令集如 x86。

同样对昇腾 AI 处理器而言,也有一套专属的指令集。昇腾 AI 处理器的指令集设计介于精简指令集和复杂指令集之间,包括了标量指令、向量指令、矩阵指令和控制指令等。标量指令类似于精简指令集,而矩阵、向量和数据搬运指令类似于复杂指令集。昇腾 AI 处理器指令集结合精简指令集和复杂指令集两者的优势,在实现单指令功能简单和速度快的同时,对于内存的操作也比较灵活,搬运较大数据块时操作简单、效率较高。

1. 标量指令

标量指令主要由标量计算单元执行,主要的目的是为向量指令和矩阵指令配置地址以及控制寄存器,并对程序执行流程进行控制。标量指令还负责输出缓冲区中数据

的存储和加载,以及进行一些简单的数据运算等。昇腾 AI 处理器中常用的标量指令如表 5-1 所示。

表 5-1 常用标量指令

类 别	指令举例
运算指令	ADD. s64 Xd, Xn, Xm
	SUB. s64 Xd, Xn, Xm
	MAX. s64 Xd, Xn, Xm
	MIN. s64 Xd, Xn, Xm
比较与选择指令	CMP. OP. type Xn, Xm
	SEL. b64 Xd, Xn, Xm
逻辑指令	AND. b64 Xd, Xn, Xm
	OR. b64 Xd, Xn, Xm
	XOR. b64 Xd, Xn, Xm
数据搬运指令	MOV Xd, Xn
	LD. type Xd, [Xn], {Xm, imm12}
	ST. type Xd, [Xn], {Xm, imm12}
流程控制指令	JUMP { #imm16, Xn}
	LOOP { #uimm16, LPCNT}

2. 向量指令

向量指令由向量计算单元执行,类似于传统的单指令多数据(Single Instruction Multiple Data, SIMD)指令,每个向量指令可以完成多个操作数的同一类型运算,可以直接操作输出缓冲区中的数据且不需要通过数据加载指令来操作向量寄存器中存储的数据。向量指令支持的数据类型为 FP16、FP32 和 INT32。向量指令支持多次迭代执行,也支持对带有间隔的向量直接进行运算。常用的向量指令如表 5-2 所示。

表 5-2 常用向量指令

类 别	指令举例
向量运算指令	VADD. type [Xd], [Xn], [Xm], Xt, MASK
	VSUB. type [Xd], [Xn], [Xm], Xt, MASK
	VMAX. type [Xd], [Xn], [Xm], Xt, MASK
	VMIN. type [Xd], [Xn], [Xm], Xt, MASK
向量比较与选择指令	VCMP. OP. type CMPMASK, [Xn], [Xm], Xt, MASK
	VSEL. type [Xd], [Xn], [Xm], Xt, MASK

续表

类 别	指 令 举 例
向量逻辑指令	VAND. type [Xd], [Xn], [Xm], Xt, MASK
	VOR. type [Xd], [Xn], [Xm], Xt, MASK
数据搬运指令	VMOV [VAd], [VAn], Xt, MASK
	MOVEV. type [Xd], Xn, Xt, MASK
专用指令	VBS16. type [Xd], [Xn], Xt
	VMS4. type [Xd], [Xn], Xt

3. 矩阵指令

矩阵指令由矩阵计算单元执行,实现高效的矩阵相乘和累加操作($C=A \times B+C$)。在神经网络计算过程中,矩阵 **A** 通常代表输入特征矩阵,矩阵 **B** 通常代表权重矩阵,矩阵 **C** 为输出特征矩阵。矩阵指令支持 INT8 和 FP16 类型的输入数据,支持 INT32、FP16 和 FP32 类型的计算。目前最常用的矩阵指令为矩阵乘加指令 MMAD,其格式如下:

MMAD. type [Xd], [Xn], [Xm], Xt

其中,[Xn],[Xm]为指定输入矩阵 **A** 和矩阵 **B** 的起始地址,[Xd]为指定输出矩阵 **C** 的起始地址。Xt 表示配置寄存器,由三个参数组成,分别为 *M*,*K* 和 *N*,用以表示矩阵 **A**、**B** 和 **C** 的大小。在矩阵计算中,系统会不断通过 MMAD 指令来实现矩阵的乘加操作,从而达到加速神经网络卷积计算的目的。

3.3 卷积加速原理

在深度神经网络中,卷积计算一直扮演着至关重要的角色。在一个多层的卷积神经网络中,卷积计算的计算量往往是决定性的,将直接影响到系统运行的实际性能。作为人工智能加速器的昇腾 AI 处理器自然也不会忽略这一点,并且从软硬件架构上都对卷积计算进行了深度的优化。

3.3.1 卷积加速

图 3-15 展示的是一个典型的卷积层计算过程,其中 X 为输入特征矩阵; W 为权重矩阵; b 为偏置值; Y_0 为中间输出; Y 为输出特征矩阵, GEMM 表示通用矩阵乘法。输入特征矩阵 X 和 W 先经过 Img2Col 展开处理后得到重构矩阵 X_{12C} 和 W_{12C} 。通过矩阵 X_{12C} 和矩阵 W_{12C} 进行矩阵相乘运算后得到中间输出矩阵 Y_0 ; 接着累加偏置 b , 得到最终输出特征矩阵 Y , 这就完成了一个卷积神经网络中的卷积层处理。

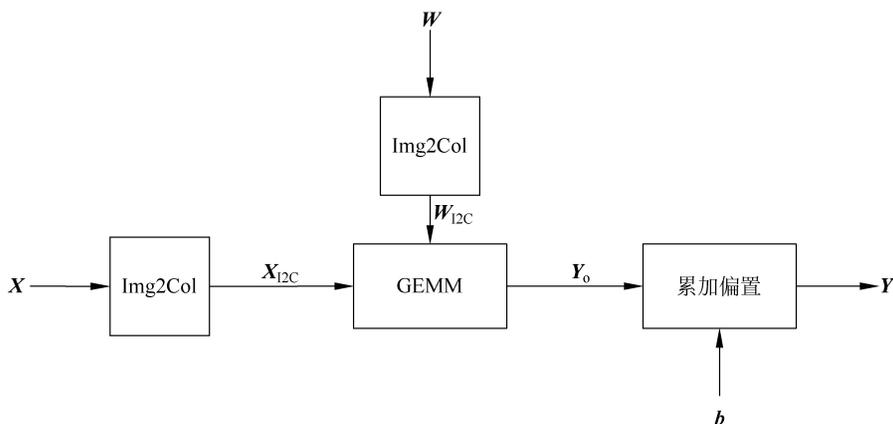


图 3-15 卷积计算过程

利用 AI Core 来加速通用卷积计算,总线接口从核外 L2 缓冲区或者直接从内存中读取卷积程序编译后的指令,送入指令缓存中,完成指令预取等操作,等待标量指令处理队列进行译码。如果标量指令处理队列当前无正在执行的指令,就会即刻读入指令缓存中的指令,并进行地址和参数配置,之后再由指令发射模块按照指令类型分别送入相应的指令队列执行。在卷积计算中首先发射的指令是数据搬运指令,该指令会被发送到存储转换队列中,再最终转发到存储转换单元中。

卷积整个数据流如图 3-16 所示,如果所有数据都在 DDR 或 HBM 中,存储转换单元接收到读取数据指令后,会将矩阵 X 和 W 由总线接口单元从核外存储器中由数据通路①读取到输入缓冲区中,并且经过数据通路③进入存储转换单元,由存储转换单元对 X 和 W 进行补零和 Img2Col 重组后得到 X_{12C} 和 W_{12C} 两个重构矩阵,从而完成卷积计算到矩阵计算的格式变换。在格式转换的过程中,存储转换队列可以发送下一个指令给存储转换单元,通知存储转换单元在矩阵转换结束后将 X_{12C} 和 W_{12C} 经过数据

通路⑤送入矩阵计算单元中等待计算。根据数据的局部性特性,在卷积过程中如果权重 W_{12C} 需要重复计算多次,可以将权重经过数据通路⑰固定在输入缓冲区中,在每次需要用到该组权重时再经过数据通路⑱传递到矩阵计算单元中。在格式转换过程中,存储转换单元还会同时将偏置数据从核外存储经由数据通路④读入到输出缓冲区中,经过数据通路⑥由存储转换单元将偏置数据从原始的向量格式重组成矩阵后,经过数据通路⑦转存入输出缓冲区中,再经过数据通路⑨存入累加器中的寄存器中,以便后续利用累加器进行偏置值累加。

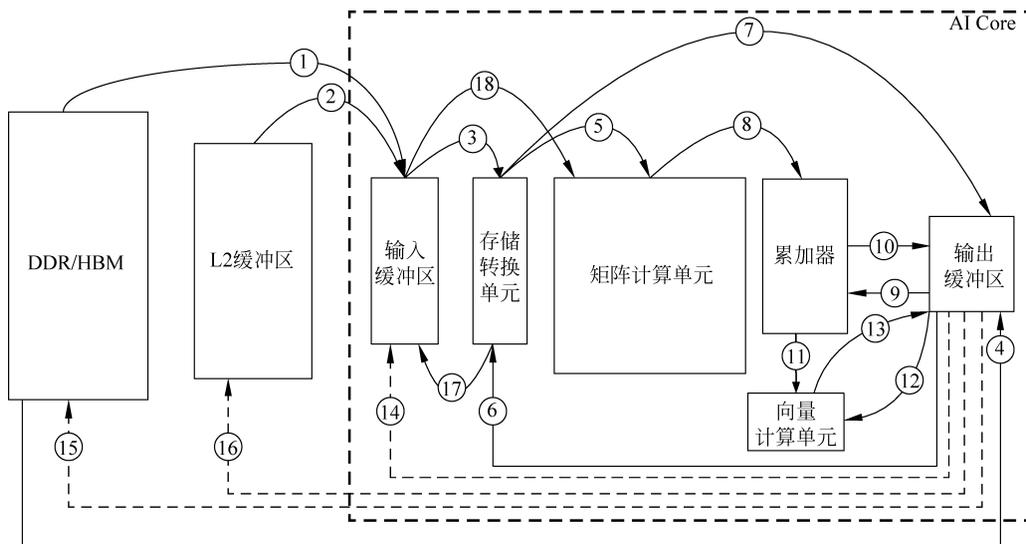


图 3-16 卷积典型数据流

当左、右矩阵数据都准备好了以后,矩阵运算队列会将矩阵相乘指令通过数据通路⑤发送给矩阵计算单元。 X_{12C} 和 W_{12C} 矩阵会被分块组合成 16×16 的矩阵,由矩阵计算单元进行矩阵乘法运算。如果输入矩阵较大,则可能会重复以上步骤多次并累加得到 Y 。中间结果矩阵,存放于矩阵计算单元中。矩阵相乘完成后,如果还需要处理偏置值,累加器会接收到偏置累加指令,并从输出缓冲区中通过数据通路⑨读入偏置值,同时经过数据通路⑧读入矩阵计算单元中的中间结果 Y 。并累加,最终得到输出特征矩阵 Y ,经过数据通路⑩被转移到输出缓冲区中等待后续指令进行处理。

AI Core 通过矩阵相乘完成了网络的卷积计算,之后向量执行单元会接收到池化和激活指令,输出特征矩阵 Y 就会经过数据通路⑫进入向量计算单元进行池化和激活处理,得到的结果 Y 会经过数据通路⑬存入输出缓冲区中。向量计算单元能够处理激

活函数等一些常见的特殊计算,并且可以高效实现降维的操作,特别适合做池化计算。在执行多层神经网络计算时, Y 会被再次从输出缓冲区经过数据通路⑭转存到输入缓冲区中,作为输入重新开始下一层网络的计算。

达芬奇架构针对通用卷积的计算特征和数据流规律,采用功能高度定制化的设计,将存储、计算和控制单元进行有效的结合,在每个模块完成独立功能的同时实现了整体的优化设计。AI Core 高效组合了矩阵计算单元与数据缓冲区,缩短了存储到计算的数据传输路径,降低延时。

同时 AI Core 在片上集成了大容量的输入缓冲区和输出缓冲区,一次可以读取并缓存充足的数据,减少了对核外存储系统的访问频次,提升了数据搬移的效率。同时,各类缓冲区相对于核外存储系统具有较高的访问速度,大量片上缓冲区的使用也极大提升了计算中实际可获得的数据带宽。

同时针对深度神经网络的结构多样性,AI Core 采用了灵活的数据通路,使得数据在片上缓冲区、核外存储系统、存储转换单元以及计算单元之间可以快速流动和切换,从而满足不同结构的深度神经网络的计算要求,使得 AI Core 对各种类型的计算具有一定的通用性。

3.3.2 架构对比

在此回顾一下前面已经阐述过的 GPU 和 TPU 处理卷积的方式,并和昇腾 AI 处理器中的达芬奇架构做一个对比。以如图 2-5 所示的进行卷积加速的案例为例,GPU 采用通用矩阵乘法方式,将卷积计算转换成擅长的矩阵计算,利用海量线程在多个时钟周期内通过并行处理得到输出特征矩阵。GPU 利用流处理器中的 SIMD 同时执行多个线程,而这些线程在一定时间内完成一个操作步骤,如矩阵乘法、激活或池化等。GPU 中线程完成了一个操作步骤之后,由于片上存储容量较小,往往需要将数据写回到内存中,直到这些线程执行下一步计算时才会再次读入内存中的数据。由于 GPU 的通用计算能力,无论是卷积计算还是池化、激活计算都可以采用类似的执行模式进行处理。

TPU 采用了脉动阵列的方式,对卷积计算进行了直接的加速。TPU 在片上设计了大容量的缓冲区,可以一次将几乎所有计算需要用到的数据都读入片上缓冲区中,之后由内部脉动数据流控制单元将数据传入脉动计算阵列中执行。在脉动阵列全部计算单元都被占满时,达到饱和状态,且每一个时钟周期都可输出一行元素的结果。

TPU 在脉动阵列中执行完卷积运算后,需要经过定制的矢量运算单元进行激活或池化处理。

昇腾 AI 处理器进行卷积计算时也采用了通用矩阵乘法的方式进行加速。它首先将输入特征矩阵和权重矩阵展开重组,就好像 GPU 一样,然后进行矩阵相乘来实现卷积计算。但是由于硬件架构和设计的不同,昇腾 AI 处理器在处理矩阵运算时和 GPU 相比存在显著的差异。由于矩阵计算单元一次可以对 16×16 大小的矩阵进行计算,并可以在很短的时间内计算出结果,所以相比 GPU 来说,昇腾 AI 处理器提高了矩阵计算的吞吐率。同时,由于向量计算单元和标量计算单元的存在,可以并行地处理卷积、池化、激活等多种计算,进一步提高了神经网络计算过程的并行性。

GPU 通过集中式的寄存器管理、多线程并行化执行、类 CPU 的片上组织结构,可以实现灵活的任务分配与调度控制,程序开发难度相对较低。但是为了满足通用性的要求所付出的代价是每次计算过程都需要符合通行的规则,比如从寄存器取数开始到结果存回寄存器结束,这样就增加了数据在搬运过程中的代价,从而使得整体计算功耗升高。

TPU 在实现卷积神经网络计算时,通过极为精简的指令一次性将大量数据读入缓冲区中,并以固定的模式流入脉动阵列进行计算,可以大大减少数据搬运的次数并降低系统整体功耗。但是这样定制化的设计限制了芯片的应用领域。同时由于片上缓冲区过大,在同等面积下,必然会压缩其他有效资源,从而限制了 TPU 的能力。

昇腾 AI 处理器具有对矩阵和卷积运算的高效性,主要得益于达芬奇架构所采用的定制化硬件设计,专门用来加速矩阵运算,能够快速完成 16×16 大小的矩阵乘法。同时达芬奇架构也采用了较多的分布式缓存与计算单元进行配合,能有效减少数据到计算单元的搬运,在提升计算力的同时还减小了数据传输的功耗,使得昇腾 AI 处理器在深度卷积神经网络这一应用领域具有一定的优势,尤其是针对大规模的卷积计算。但是由于达芬奇架构采用了固定的 16×16 大小的矩阵计算单元,所以在处理规模较小的网络时,容易造成矩阵计算单元算力难以发挥、硬件资源利用率低下的问题。

综上所述,无论哪一种架构都有其独特的优势,也存在劣势。脱离其特定的应用领域来判断一个硬件架构的好坏是没有意义的。即便是在某一应用领域内,数据的大小和分布、计算的类型和精度、网络的结构和设置等都会极大地影响硬件架构优势的

发挥。

不同于传统意义上的专用芯片,对于未来行业发展越来越重要的特定域架构芯片来说,硬件设计只是影响最终性能的一个维度,而它内在能力的发挥很大程度要取决于运行其上的软件的效率。在本书后续章节中将着重介绍软件优化的方法和案例,目的就是希望帮助读者建立软硬件协同编程的理念,从而可以真正适应后摩尔时代计算机行业发展的需求。