**2025年中国研究生数学建模竞赛A题**

通用神经网络处理器下的核内调度问题

1. 背景

在如今各类通用神经网络加速处理器（Neural Processing Unit，NPU）中，基于单指令多数据流（Single Instruction Multiple Data，SIMD）架构的处理器硬件设计简单，面效高[[1]](#footnote-1)，成为边缘推理任务的首选，但其软件模型适配的复杂性成为其大规模商用的关键瓶颈。在神经网络推理过程中，算子（如矩阵乘Matmul、卷积Conv、注意力Attention等）是最小任务单元，其执行效率直接影响模型在平台上端到端的推理性能。我们将算子在SIMD架构硬件平台上的完整计算过程拆解为由硬件单元操作构成的细粒度计算图，并通过手工或自动的方式编排成可在SIMD平台[[2]](#footnote-2)上执行的任务。由于这类计算图具有高度异构性（算子类型多样、输入形状动态变化、拓扑结构复杂），人工编排方式难度大，缺乏通用性，且效率低下，无法在如今日益复杂的计算场景下大规模推广。

因此，亟需设计一种通用调度算法，自动地将计算图中各原子操作编排调度到各硬件单元上执行，取代低效的人工编排计算单元流水的过程。该算法需面向SIMD平台的硬件限制，给出由硬件单元操作组成的计算图的优化调度顺序，使得多个不同单元能并行执行，缩短流水线延迟。同时，调度算法还需考虑内存与计算协同，动态管理多级缓存，自动决定缓存数据换入换出，减少数据搬运开销。

1. 核内调度算法
   1. 任务

计算图为有向无环图（DAG），本题开始给出的图，其节点分为两类：在特定硬件单元上执行的操作节点，以及用于缓存资源分配的缓存管理节点。给定的一个计算图，需要通过调度算法提供一个包含计算图所有节点的有序列表作为调度顺序，以及为所有缓存管理节点中的申请节点分配相应类型的地址。在本核内调度问题中，处理核心由多个执行单元和多级缓存共同组成（参考附录A）。

计算图中的每个操作节点具有如下关键属性：

* **Id**：从0开始的节点唯一标识符。
* **Op**：操作指令名。可以是除ALLOC/FREE以外的任意操作名（例如数据搬运类操作COPY\_IN，COPY\_OUT等，计算类操作ADD，MUL等）。
* **Pipe**：指定该操作在哪个执行单元上运行。如附录A图3中，Cube、Vector为计算单元，MTE1/2/3/FIXP为数据搬运单元。
* **Cycles**：表示在该执行单元上操作运行所需的时钟周期数。
* **Bufs**：表示该操作所需输入和输出数据缓冲区的唯一标识符（BufId）组成的列表。

计算图中的每个缓存管理节点具有如下关键属性：

* **Id**：从0开始的节点唯一标识符。
* **Op**：ALLOC或FREE，分别代表申请或释放数据缓冲区。
* **BufId**：所申请或释放的数据缓冲区的唯一标识符。
* **Size**：申请或释放的数据缓冲区长度。
* **Type**：该数据缓冲区所处的缓存类型（如附录A图3中L1/UB/L0A/L0B/L0C等）。

计算图中的每条有向边（下文中也称之为依赖边）表示源节点和目的节点之间的执行依赖关系，节点必须严格按照依赖边约束的前后顺序执行。

图1为一个示例Exp计算图，其表现出计算图如下基本规则：

* 首先，忽略缓存管理节点后，所有的计算图都以COPY\_IN节点开始（将数据从核外内存搬入核内缓存），经过一系列核内基本运算，最终以COPY\_OUT节点结束（将计算结果从核内缓存搬出至核外内存）。
* 其次，所有操作节点的执行都受事先分配好的缓存资源限制。因此，计算图中所有根节点均为ALLOC节点，所有叶子节点均为FREE节点，反之亦然。
* 最后，ALLOC节点发出的有向边通常指向对应缓冲区数据的生产者（例如，Id为0的ALLOC节点指向COPY\_IN节点）；而该缓冲区内数据的消费者则通过有向边指向FREE节点（例如，EXP节点指向Id为4的FREE节点）；生产者和消费者之间也通过有向边相连。

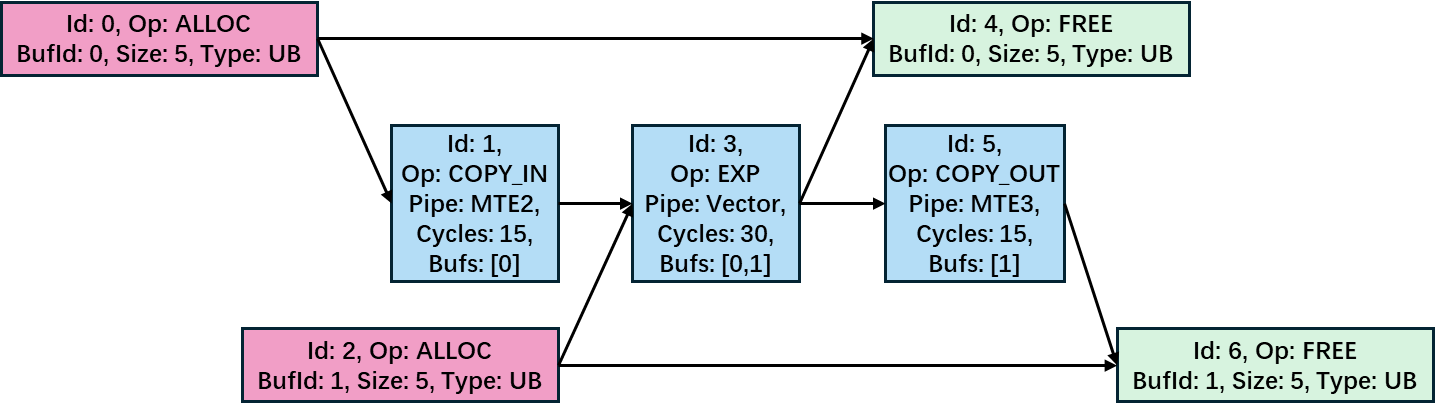


图1 计算图示例

调度算法的首要任务是给出一个满足计算图拓扑序[[3]](#footnote-3)的节点执行序列，该序列应该包含计算图中所有节点（该序列由节点Id组成）。以图1所示的示例Exp计算图为例，调度序列（0 → 1 → 2 → 3 → 4 → 5 → 6）为一个有效的调度序列，因为该序列包含了所有节点，且满足拓扑序。

调度算法需要为计算图中每个数据缓冲区（即每个BufId）分配一个起始地址偏移（Offset），以确定其在硬件缓存中所占的位置。硬件上的各类缓存均为有限长度的连续地址空间，每个缓冲区需在指定缓存空间中占据一片连续的地址区间，其区间长度由Size字段指定。不同数据缓冲区可以分时复用同一片区间，但同一时刻存在的缓冲区，必须分配在不重叠地址区间。

对于图1中的示例计算图，Buf0（BufId为0的缓冲区）和Buf1在给出的调度序列上生命周期[[4]](#footnote-4)重叠，因此需分配至不重叠的地址区间。Buf0和Buf1的长度都为5，若UB类型缓存容量为10，则可为Buf0分配[0-4]的地址区间，为Buf1分配[5-9]的地址区间，因此缓存地址分配方案{Buf0:Offset=0,Buf1:Offset=5}是可行的。

这样就得到了一个包括节点调度顺序和缓存地址分配在内的完整调度方案，其在实际硬件上的执行过程如图2所示。

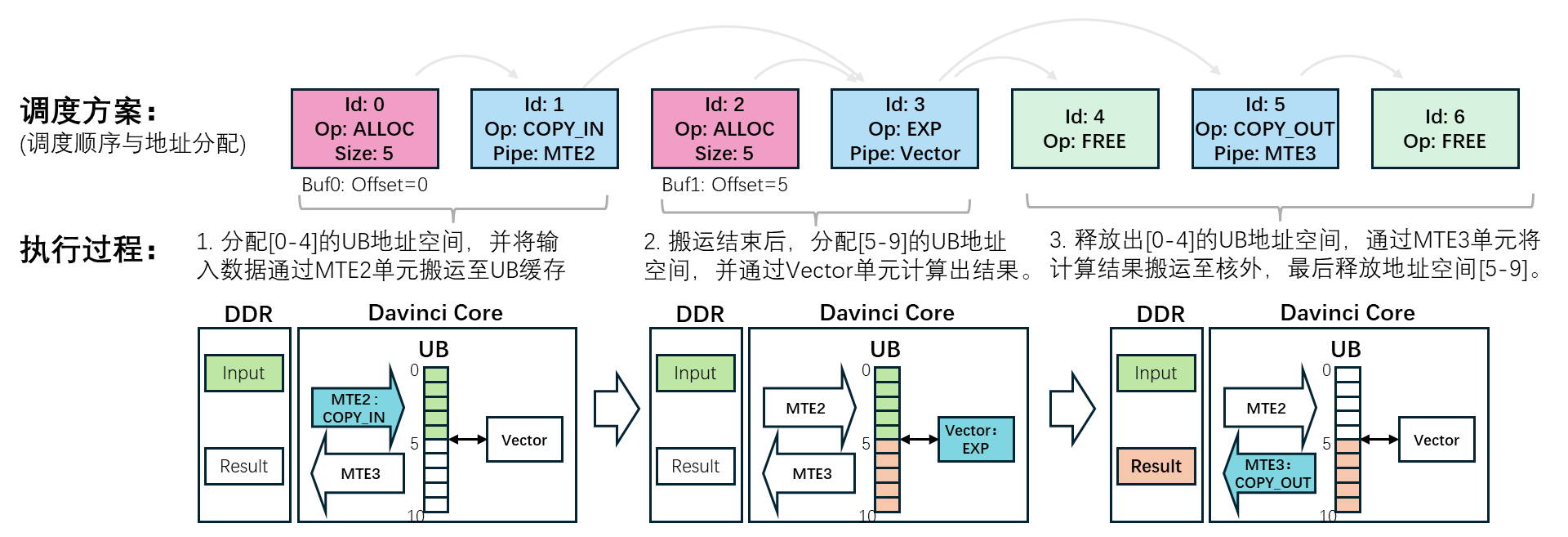


图2 硬件执行过程示意图

由于硬件缓存大小有限，调度算法可能会遇到空间不足的问题：在执行中的某一个时刻，需要驻留数个缓冲区，然而硬件空间大小可能不满足这些数据缓冲区的需要，无法生成可行的调度方案。此时，调度算法需要在计算图中加入数据搬运操作节点，将部分核内缓冲区数据临时移出至核外内存（Double Data Rate SDRAM，简称DDR）上，释放出硬件缓存空间，并在后续计算需要时，重新载入至核内缓存。上述操作称为缓存换入换出操作（简称SPILL操作），具体给出方式见附录B。



* 1. 算法评估指标

评估调度算法的性能主要依据两个关键指标：任务的总执行时间和总额外数据搬运量。

* + 1. 总执行时间

总执行时间表示在硬件上执行完计算图需要的总时钟周期数，用于衡量调度方案在硬件上的实际运行效率。硬件上各个不同单元虽然可以同时工作，但由于指令间存在依赖关系，某条指令要能在某个单元上执行，必须等待其前序指令（在其他单元上）执行完毕。两种依赖关系可能导致节点执行时需要等待：一是计算图中节点间已有的依赖边，二是缓存物理地址复用带来的执行依赖[[5]](#footnote-5)。调度方案下的总执行时间根据调度顺序与依赖关系确定（具体计算方法请参考附录C）。

优良的调度方案能使得各执行单元交替工作，并行地流水化处理计算图各节点，最小化各执行单元等待时间，从而减少总执行时间，提升运行效率。

* + 1. 总额外数据搬运量

总额外数据搬运量指由缓存换入换出操作（SPILL操作）所引起的额外DDR访问数据量。DDR访问通常伴随较高功耗，该值越低，代表计算图在平台上执行的能效越高。具体计算方法详见附录C。

调度方案需要尽可能避免资源不够时产生的缓存换入换出操作。一方面，调度需要减少缓存碎片化，尽可能地在有限空间内放下更多的缓存数据；另一方面，调度需要尽可能优化调度顺序，压缩每个缓冲区数据的生命周期，避免缓冲区数据持久驻留在缓存。

1. 最小缓存驻留调度

对于某个计算图，每个节点需要的缓存大小记为：

一旦给出一个节点调度顺序，就可直接计算得到该序列在执行过程中需要驻留的最大缓存容量：令初始驻留缓存容量，按顺序遍历各节点；若遇到ALLOC节点，则增加相应大小，若遇到FREE节点，则减小相应大小。在此过程中，的最大值即为需要驻留的最大缓存容量（）[[6]](#footnote-6)。

为使得调度过程需要驻留的缓存尽量不 超过硬件限制（实际调度过程中尽量避免缓存换入换出），需要寻找一种优良的调度序列，使得执行过程所需驻留的最大缓存容量尽可能小。

不考虑实际硬件上L1和UB缓存的长度限制，请设计一种方案，获取使得尽可能小的调度序列（本问题中仅关注调度顺序，不涉及缓存分配策略），同时，请分析算法相对于节点总数的时间复杂度。

附件中提供了六个示例计算图（文件格式参见附录E），请设计调度算法方案，给出所有示例计算图上的结果（一定要放在论文内），并且以附件形式提供各计算图的调度顺序结果（文件格式见附录E）。

1. 缓存分配与换入换出

基于问题1得到的调度顺序，为计算图中各缓冲区分配实际的物理地址偏移。

假设硬件缓存容量的限制如表1，在调度序列的基础上，设计一种使得总额外数据搬运量指标尽可能小的缓存分配方案。方案需给出计算图中每个缓冲区对应的地址偏移、分配过程需要的SPILL操作列表，以及加入SPILL节点后的完整调度序列。可从减少缓存碎片的角度分析如何优化最终结果。若以问题1中调度序列结果为基础进行缓存分配，无法充分实现总额外数据搬运量最小化的目标，可进一步对调度序列生成策略进行优化调整。

|  |  |  |  |
| --- | --- | --- | --- |
| 缓存 | 容量 | 缓存 | 容量 |
| L1 | 4096 | **L0B** | 256 |
| UB | 1024 | **L0C** | 512 |
| L0A | 256 |  |  |

表1 硬件缓存资源大小列表

参赛论文中需提供附件中示例计算图上的总额外数据搬运量结果（一定要放在论文内），并在提交附件中提供这些示例计算图上的详细调度结果（格式见附录E）。

1. 性能优化策略

结合问题1得到的调度顺序和问题2得到的缓存分配方案，已经可以组成一套完整的调度算法，但由于前两个问题的优化目标围绕减少数据搬运量展开，因此问题2得到的总运行时间指标可能不理想。

请在总额外数据搬运量指标不显著增加的前提下，进一步设计优化策略，尽可能降低算子的总运行时间（也可对两项指标联合优化）。可以从如何优化调度顺序入手，也可以从如何优化缓存分配方案入手。实现其设计的性能优化策略，在参赛论文中给出附件中示例计算图上两项主要指标数据的提升效果（一定要放在论文内），并以附件的形式提供优化后的详细调度结果（格式见附录E）。

附录

1. 架构抽象模型

在本核内调度问题中，我们将SIMD架构中的处理核心抽象为多个可并行工作的执行单元与多级缓存共同组成的计算结构，如图3所示。执行单元主要包括计算单元（Cube核与Vector核）以及数据搬运单元（MTE1/2/3和FIXP）。每个单元具备独立的指令队列，由统一的调度核心进行指令分发与协同调度。各单元功能如下：

**数据搬运单元** (包括MTE1/2/3单元和FIXP单元)：负责在核外内存（DDR）、多级缓存（L1, L0A/L0B/L0C）以及统一缓存（UB）之间传输数据。

**计算单元** (包括Cube核和Vector核)：

* + **Cube核**: 专用于矩阵乘法计算。输入数据（左矩阵A、右矩阵B）经由MTE2（DDR→L1）和MTE1（L1→L0）加载至专用的L0A与L0B缓存。计算结果写入L0C缓存，最后由FIXP单元写回DDR。
  + **Vector核**: 负责通用向量计算。输入数据和中间结果都存储在统一缓存（UB）内，输入数据通过MTE2单元从DDR加载至UB，最终运算结果则通过MTE3单元从UB写回DDR。

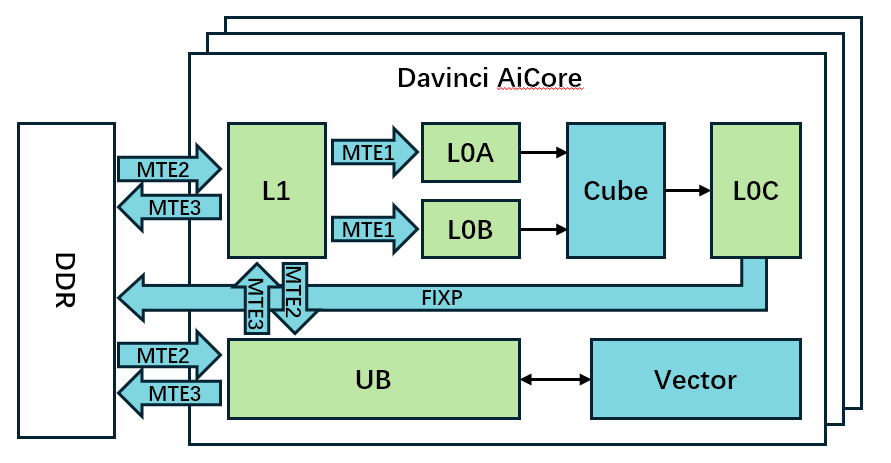


图3 Davinci架构抽象图

核心内设有多级缓存：L0缓存为紧邻Cube核的高速缓存，分为L0A（左矩阵输入）、L0B（右矩阵输入）、L0C（结果输出）；L1缓存容量更大，作为矩阵运算数据的中间缓存；统一缓存UB则用于存储Vector核的输入、输出及中间结果。MTE单元也支持在L1和UB之间交换数据，以支持矩阵计算和向量计算交替执行的场景。

在本核内调度问题中，计算图的每个节点可视为一条在特定单元上执行的指令。调度算法的首要任务是为所有节点（指令）生成一个满足计算图节点依赖关系的有效拓扑序。基于该序列中的指令间依赖关系，系统将自动插入同步指令，并由调度核心分发指令流，驱动各单元协同完成计算任务（该过程由框架自动实现，无需调度算法处理）。

此外，调度算法需管理缓存资源的使用。每种缓存资源可以看作是连续的地址空间，且有固定的总长度，行为完全由程序控制。因此，调度算法需为计算图中每个缓冲区分配指定类型与大小的缓存地址区间。

1. 缓存换入换出机制说明

当调度算法遇到缓存资源死锁时（即：缓存空间已分配或接近分配完、已分配缓冲区因依赖指令未完成而无法释放，同时新指令的执行需要分配缓存），需通过缓存数据换入换出操作（SPILL操作）临时将部分缓冲区数据移至核心外的内存DDR上，释放出这部分缓存空间，并在后续计算需要时再搬移回来。我们将硬件上缓存数据搬出和搬入的过程，作为两个新增的特殊操作节点：SPILL\_OUT和SPILL\_IN。

如图4所示，若引入一次SPILL操作，并指定某一缓冲区（如图4中Buf0）作为操作对象，会向计算图中插入两个操作节点：

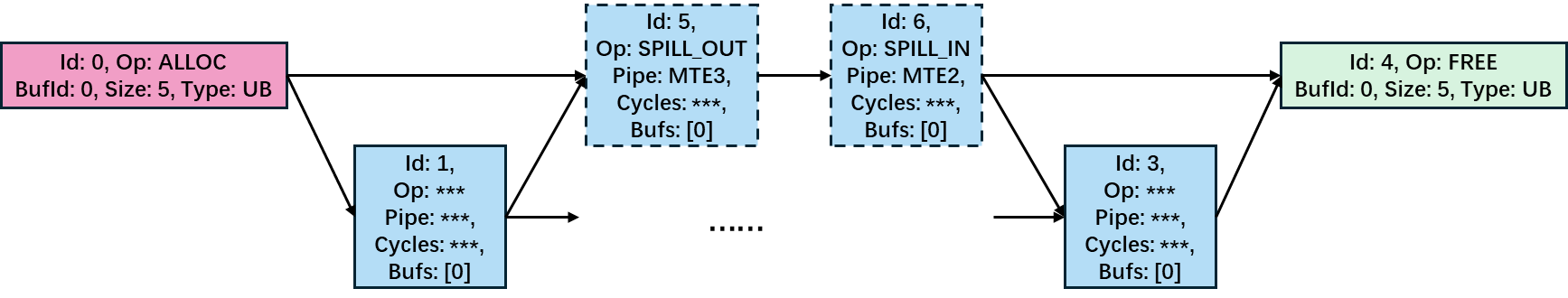


图4 缓存数据换入换出示意图

SPILL\_OUT节点将当前Buf0的数据搬移至DDR，SPILL\_IN节点再将其重新搬回UB缓存。这两个节点分别需要占用MTE3单元和MTE2单元的执行时间。

对于新增的操作节点，其属性自动设置如下：

* **Id**：从开始自动递增。若原计算图共有个节点，Id范围为至。首次SPILL操作对应的SPILL\_OUT和SPILL\_IN节点Id分别为和。以此类推，第次SPILL操作引入的节点Id依次为和。
* **Op**：SPILL\_OUT或SPILL\_IN。
* **Pipe**：SPILL\_OUT在MTE3单元上执行，SPILL\_IN在MTE2单元上执行。
* **Cycles**：执行周期与当前目标缓冲区的大小有关，计算公式参考附录D。
* **Bufs**：仅包含当前目标缓冲区BufId的列表。

若调度算法给出次SPILL操作，则最终调度序列必须包含所有原计算图节点和新增的个SPILL操作节点。

新增节点的依赖边按如下规则生成：首先，生成三条基本依赖边：ALLOC → SPILL\_OUT、SPILL\_OUT → SPILL\_IN、SPILL\_IN → FREE。其次，将当前计算图中所有使用到目标缓冲区的操作节点分为“已执行”和“未执行”两部分（以SPILL\_OUT节点在调度序列中的位置为界）。对于“已执行”的操作节点，生成 → SPILL\_OUT的依赖边；对于“未执行”的操作节点，生成SPILL\_IN → 的依赖边。

由于SPILL\_IN节点可将数据加载至新的核内缓存地址区间，调度算法需明确给出每次操作的目标缓冲区以及新的目标地址偏移。具体表示方式参考附录E。

1. 调度算法评估指标详细说明
2. 总执行时间

根据调度算法给出的执行顺序、缓存分配方案、缓存换入换出操作列表，可以通过如下步骤计算出总执行时间：

**步骤1：插入SPILL节点**。依照附录B所述，在原计算图中插入SPILL\_OUT节点和SPILL\_IN节点，并添加相应的依赖边（若不需要缓存换入换出，则跳过该步骤）。

**步骤2：计算缓存复用依赖**。按ALLOC执行顺序遍历各缓冲区，检查其是否复用了之前已分配缓冲区的地址。若缓冲区b复用了缓冲区a的地址空间，则需要添加一条从a的FREE节点到b的ALLOC节点的依赖边[[7]](#footnote-7)。

**步骤3：流水排布**。按步骤1和2修改得到计算图，每个节点的起始时间和结束时间（单位为时钟周期）可根据以下约束依次计算得到：

* 时间非负：。
* 资源独占：同一执行单元内，任意时刻只能有一个节点在执行。[[8]](#footnote-8)
* 执行依赖：节点的起始时间大于或等于其所有前序节点的结束时间。
* 执行耗时：。若为缓存管理节点，则。

任务总执行时间定义为所有节点结束时间的最大值。

步骤3的计算逻辑可通过流水图进行可视化，以更直观地呈现整个算子的执行过程。流水图用于展示算子执行过程中各硬件单元的工作状态，其横轴代表时间，纵轴代表不同的执行单元。以图1中的Exp计算图为例，按0至6的调度顺序，可得到图5所示的时序流水图：节点0、2、4、6为缓存管理指令，未予显示；节点1、3、5则分别在对应单元上执行。由于存在1→3、3→5的依赖关系，这些节点在时序上必须串行执行，无法重叠。图中可看出，该算子总执行时间为60个时钟周期。

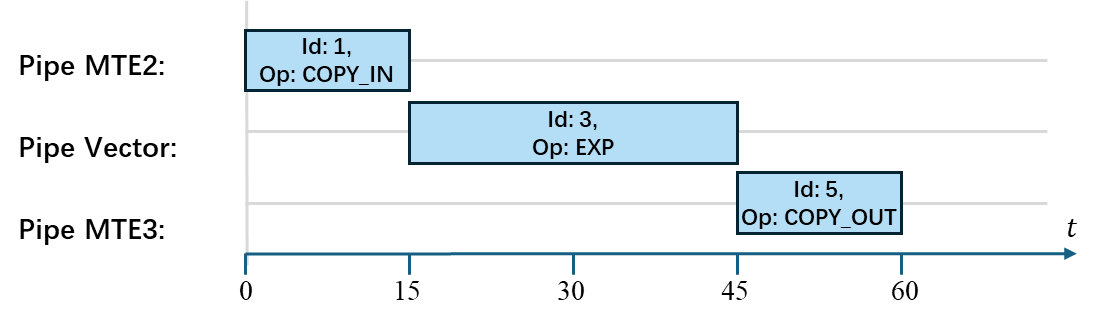


图5 调度顺序时序流水图

不同硬件单元可以并行执行。若计算图中存在多个Exp子计算图，在总缓存容量允许的情况下，后续COPY\_IN节点可提前执行（图6中节点8、节点15），从而实现多个执行单元间的流水并行，减少总执行时间。

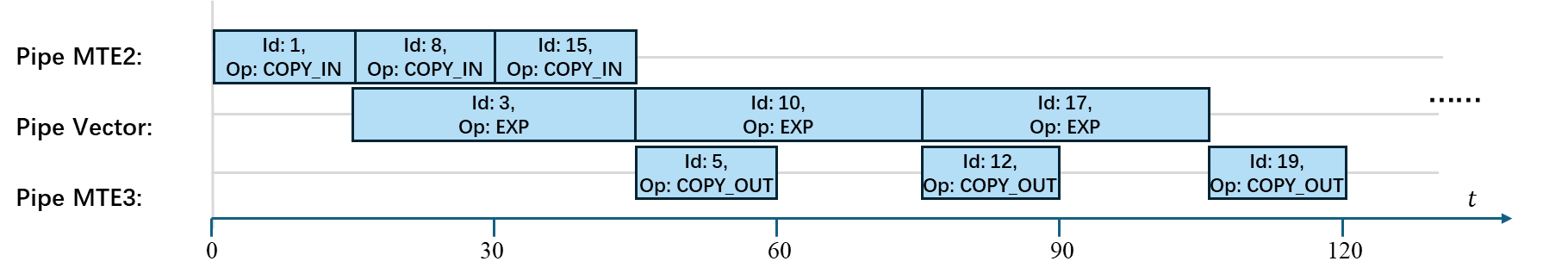


图6 调度顺序时序流水图

1. 总额外数据搬运量

总额外数据搬运量根据调度算法给出的SPILL操作列表中的目标缓冲区进行统计。对不同的目标缓冲区，按如下两种情况分别计算：

情况1：若目标缓冲区在计算图中未被任何COPY\_IN节点使用（即所有COPY\_IN节点的Bufs属性列表中均不包含该缓冲区），则此次SPILL操作将引入两次核外DDR存取：SPILL\_OUT和SPILL\_IN分别执行一次数据搬出与搬入。因此，增加的额外数据搬运量为该缓冲区大小的两倍。

情况2：若目标缓冲区在计算图中被某个COPY\_IN节点使用，则仅SPILL\_IN节点需要执行一次从DDR至核内的数据搬入，SPILL\_OUT不产生实际核外存取。因此，增加的额外数据搬运量等于该缓冲区大小。

总额外数据搬运量为所有SPILL操作所带来的额外数据搬运量之和。

1. 缓存数据换入换出操作耗时计算

对于每个SPILL\_OUT操作和SPILL\_IN操作，其消耗的时间跟目标缓冲区大小成线性相关。对不同的目标缓冲区，按如下两种情况分别计算：

情况1：若目标缓冲区在计算图中未被任何COPY\_IN节点使用，则按如下公式计算耗时：

情况2：若目标缓冲区在计算图中被某个COPY\_IN节点使用（该节点仅可能存在一个），则SPILL\_OUT节点耗时为0，SPILL\_IN节点耗时按上述公式计算。

1. 示例计算图数据格式及提交附件格式说明

本赛题提供一系列示例计算图，以供参赛队伍验证针对设置问题所提出解决方案的效果。这些示例计算图的任务名以及节点数量等特性列在表2中。

需要说明的是，这一系列示例计算图仅提供给参赛队伍作思路验证使用。实际场景下的计算图可能是任意多个算子的排列组合，因此，参赛队伍提出的解题思路应尽可能具有泛化性与通用性，尽可能避免针对某一特定算子的计算图结构（附录F中所示的典型算子计算图结构）进行优化。

|  |  |  |
| --- | --- | --- |
| 任务名 | 节点数量 | 依赖边数量 |
| Matmul\_Case0 | 4160 | 7104 |
| Matmul\_Case1 | 30976 | 55040 |
| FlashAttention\_Case0 | 1716 | 2712 |
| FlashAttention\_Case1 | 6952 | 11184 |
| Conv\_Case0 | 2580 | 3869 |
| Conv\_Case1 | 36086 | 85653 |

表2 示例计算图及其特性列表

上述示例计算图提供Json和CSV两种版本。对于Json版本，表中每个计算图以**<任务名>.json**文本文件的形式提供。每个文件中包含内容的格式如下：

**<任务名>.json:**

{

    "Nodes" : [

        {

            "Id" : 0,

            "Op" : "ALLOC",

            "BufId" : 0,

            "Size" : 1,

            "Type" : "UB"

        },

        {

            "Id" : 1,

            "Op" : "COPY\_IN",

            "Pipe" : "MTE2",

            "Cycles" : 15,

            "Bufs" : [0]

        },

        ......

    ],

    "Edges" : [

        [0, 1],

        ......

    ]

}

其中，“Nodes”字段下为包含计算图所有操作节点和缓存管理节点的列表，每个元素为一个节点；“Edges”字段下为包含计算图所有有向依赖边信息的列表，其每条有向边通过“[源节点Id，目标节点Id]”表示。

对于CSV版本，每个计算图的节点和有向边信息分别通过**<任务名>\_Nodes.csv**和**<任务名>\_Edges.csv**文本文件提供，其包含内容格式如下：

**<任务名>\_Nodes.csv:**

Id,Op,BufId,Size,Type,Pipe,Cycles,Bufs

0,ALLOC,0,1,UB,,,

1,COPY\_IN,,,,MTE2,15,"0"

……

**<任务名>\_Edges.csv:**

StartNodeId,EndNodeId

0,1

……

CSV文件中，第一行为表头信息，后续每一行代表一个节点或一条有向边。Json版本与CSV版本计算图仅文件格式不同，其包含的计算图信息是完全相同的。

问题1至3均需提供算法运行结果作为附件。附件压缩包需组织成如下形式：

**Attachment.rar**

├── Problem1

│ └── <任务名>\_schedule.txt

├── Problem2

│ ├── <任务名>\_schedule.txt

│ ├── <任务名>\_memory.txt

│ └── <任务名>\_spill.txt

└── Problem3

├── <任务名>\_schedule.txt

├── <任务名>\_memory.txt

└── <任务名>\_spill.txt

在该压缩包中，问题1至3的结果分为三个目录存放。问题1中，仅需提供调度序列结果。问题2和问题3需要提供完整调度结果，包括调度序列、缓存分配、换入换出操作列表。每个目录下各个文件格式如下：

1. <任务名>\_schedule.txt文件为该计算图上给出的调度顺序结果，以文本形式按序给出各个节点Id，节点Id间通过换行符隔开：

**<任务名>\_schedule.txt:**

0

2

4

1

....

1. <任务名>\_memory.txt文件为该计算图上给出的缓存分配结果，以“BufId:Offset”的形式给出各个缓存分配的地址偏移，各BufId间通过换行符隔开：

**<任务名>\_memory.txt:**

0:0

1:5

2:30

3:10

....

1. <任务名>\_spill.txt文件为该计算图上给出的SPILL操作，以“BufId:NewOffset”的形式依次给出各个SPILL操作，各操作间通过换行符隔开（若不需要SPILL操作，则提供空文件）：

**<任务名>\_spill.txt:**

5:400

9:20

1:40

....

参赛队伍需将上述结果打包并压缩成rar文件作为附件单独上传。附件文件名统一格式：**Ａ<队号>.rar**。

1. 典型算子计算图结构及说明

F.1 Matmul计算图结构

图7展示了硬件完成矩阵乘法的典型计算图结构。对于大尺寸矩阵乘法（左矩阵尺寸为，右矩阵尺寸为），由于矩阵和通常无法一次性载入至L1缓存，需采用分块加载并计算的策略。如图所示：将矩阵沿轴切分为块，将矩阵沿轴切分为块（轴切分未展示），则结果矩阵的计算被分解为次小块矩阵乘法。

矩阵中每个分块的计算可归纳为四个阶段：首先通过MTE2单元执行COPY\_IN操作，将矩阵和的对应数据分块从DDR载入L1缓存；随后由MTE1单元执行MOVE操作，将数据从L1缓存搬运至L0A/L0B缓存；再由Cube单元执行MMAD操作，完成小块矩阵乘并将结果写入L0C缓存；最后通过MTE3单元执行COPY\_OUT指令，将计算结果写回DDR。

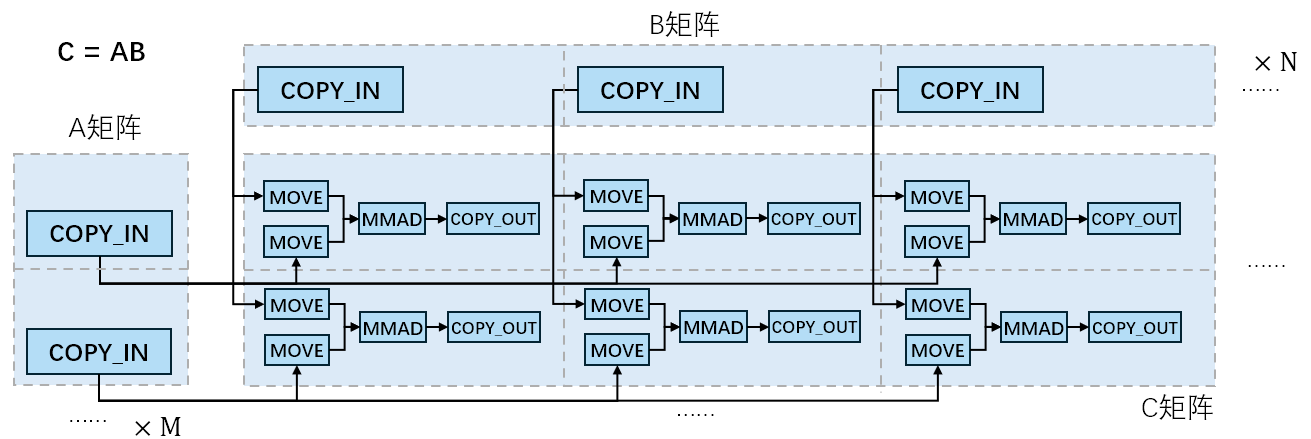


图7 Matmul计算图结构示意图

通过将不同矩阵分块间的四个计算阶段交错排列（如图8所示调度顺序），可实现各执行单元间的流水并行，提升整体计算效率。

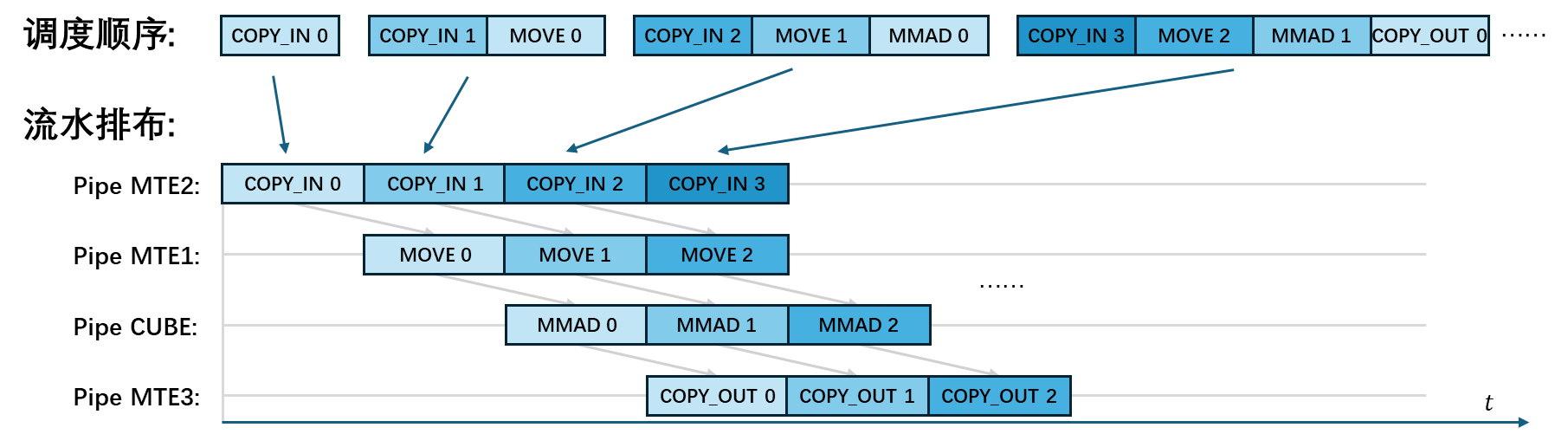


图8 Matmul分块调度与流水并行图

由于L1缓存无法容纳矩阵和的所有分块，计算过程中必然涉及矩阵分块的重复载入（通过SPILL机制实现），而矩阵分块的计算顺序将直接影响额外数据搬运量。图9展示了两种计算顺序，(a)顺序按行计算矩阵分块，(b)顺序按之字形路径计算。

假设L1缓存仅能容纳4个矩阵块，可以看出：在顺序(a)中，每个A矩阵块只需载入一次，可重复利用于整行计算；但在行间切换时，B矩阵块需重新载入（通过SPILL机制），导致B矩阵总共需载入四次。因此，顺序(a)带来的额外数据搬运量相当于矩阵B大小的三倍。同理，可以得到顺序(b)中B矩阵总共仅需载入两次，其带来的额外数据搬运量仅相当于矩阵B大小的一倍。因此，顺序(b)优于顺序(a)。

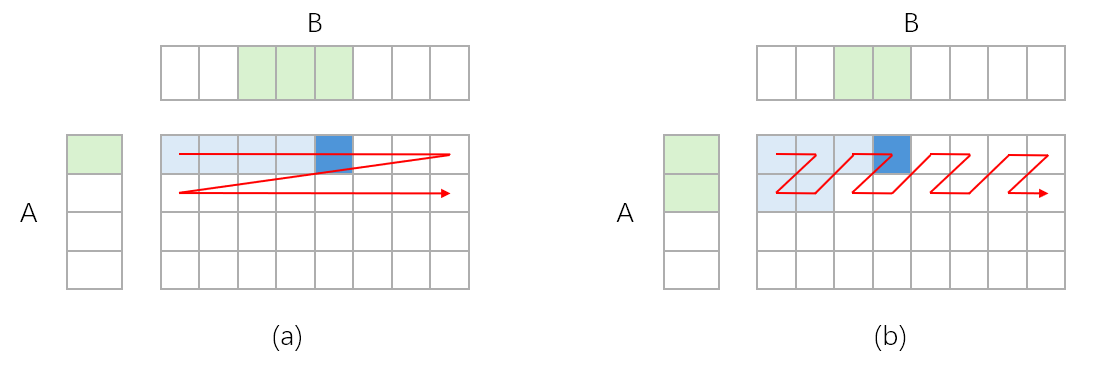


图9 矩阵乘不同调度顺序示意图

F.2 FlashAttention计算结构图

FlashAttention（FA）是一种高效计算Transformer网络中注意力机制的算法。它通过巧妙的分块计算和在线softmax技巧，将中间结果保留在高速缓存中，在保持数学等价性的同时，大幅提升了计算速度并降低了内存访问量。该算法已成为大型语言模型推理加速的关键技术之一。FA典型计算图结构如图10所示，其整体与Matmul类似，但存在如下区别：

1. 分块内部计算结构更为复杂：FA中的每个分块计算包含四个部分：Matmul、Softmax、Matmul和PostUpdate，每个部分均为一个结构较为复杂的子计算图（图中未展示细节）。其中，Matmul部分主要依靠Cube计算单元完成，而Softmax和 PostUpdate部分则主要依赖Vector计算单元实现。
2. 输出方式不同：与Matmul中每个分块独立输出不同，FA 中每一行内的所有分块共同输出一个矩阵块。具体而言，每一行前面分块的计算结果会与同一行后面分块进行加权融合，最终每行输出的是一个合并后的结果。

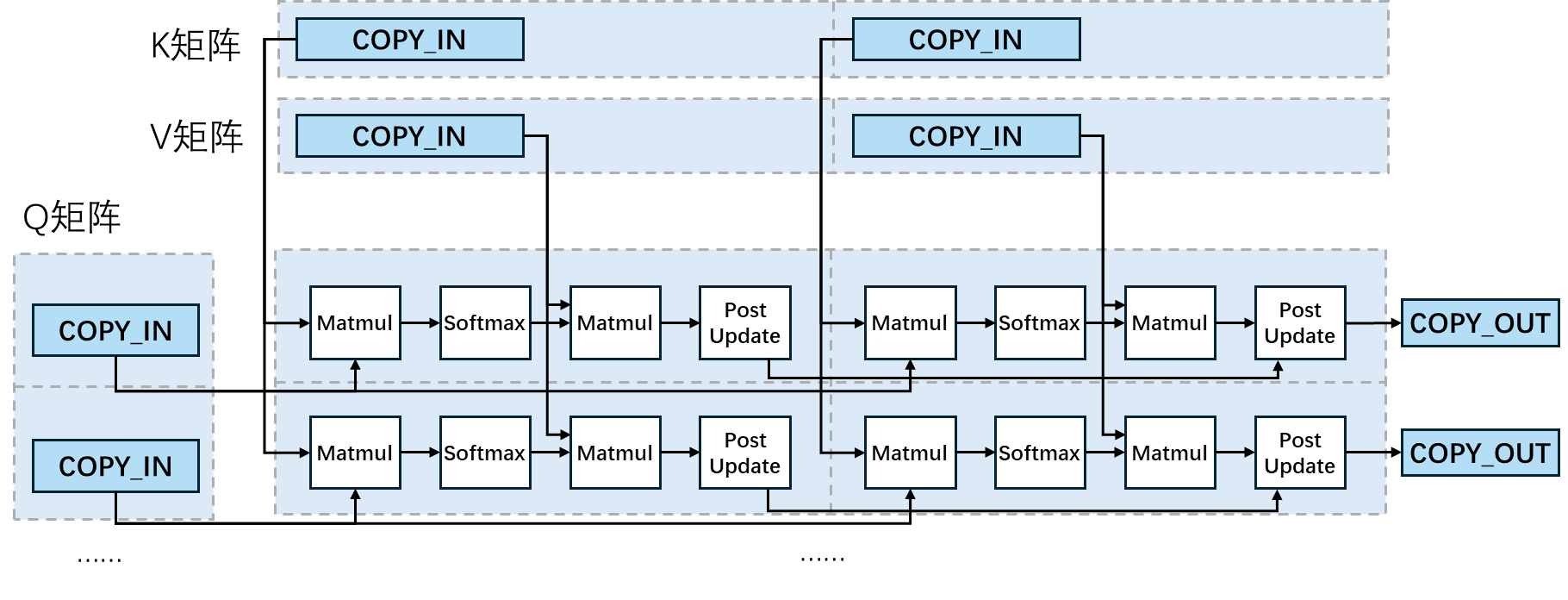


图10 FlashAttention计算图结构示意图

不同子计算图之间的中间数据结果通过MTE2和MTE3单元在UB缓存和L1缓存之间进行传输。由于每个部分同时包含Cube类计算和Vector类计算，在调度FA计算图时，需考虑Cube与Vector计算单元之间的流水并行，以及 QKV 矩阵的数据复用情况。一般而言，通过两个分块四个阶段间的交替执行，来最大化Cube单元和Vector单元的利用率，如图11所示。

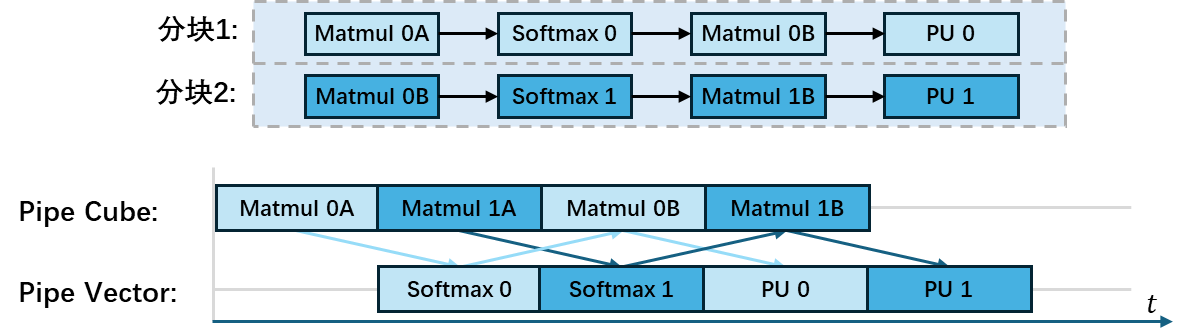


图11 FlashAttention调度示意图

F.3 卷积计算结构图

卷积神经网络（CNN）是一种专用于处理网格结构数据（如图像）的深度学习架构。CNN核心在于其多层卷积结构。通过堆叠多个卷积层，网络能够逐层提取并组合输入数据中的特征：浅层卷积捕获局部细节（如边缘和纹理），深层卷积则将这些基础特征融合成更加抽象的高级语义特征。CNN已成为计算机视觉领域最为基础和广泛的骨干网络之一。

如图12所示，一个典型的多层卷积网络计算图以前端特征图载入缓存为起点，数据被切分为多个小块载入。每一卷积层由卷积核载入操作和多个卷积操作（CONV）构成，每个卷积操作的输入包括前一层输出的多个特征块与当前层的卷积核。输出结果继续流向后续卷积层作为输入，通过不断重复这一卷积过程，最终完成对整个输入数据的特征提取与推理。

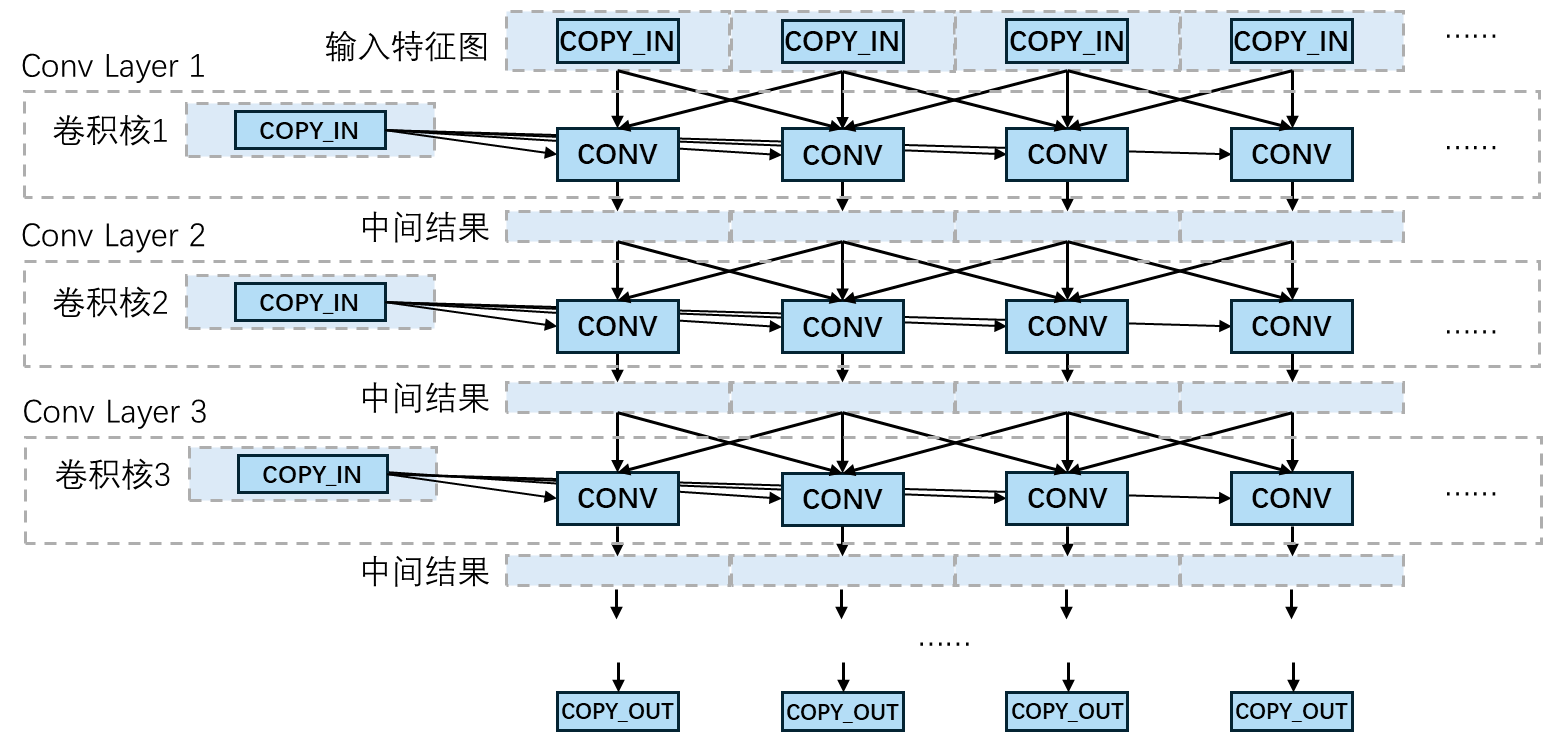


图12 卷积计算图结构示意图

卷积计算图在层间呈现交错的依赖结构，因此调度顺序对缓存驻留影响显著。通常可采用深度优先与广度优先两种调度策略，分别适用于不同场景。如图13所示，(a)和(b)分别表示深度优先和广度优先策略，图中通过节点颜色深浅表示调度顺序先后。

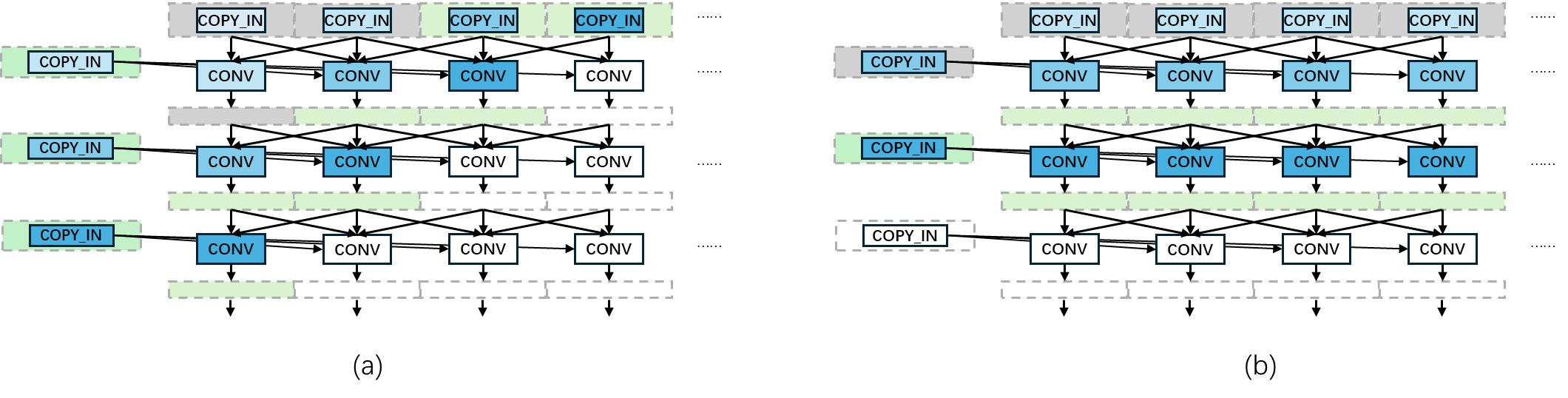


图13 卷积计算调度顺序示意图

在深度优先策略中，不同层的卷积操作交替执行，每层仅需将少量数据（图中绿色部分）驻留于缓存，灰色部分为可释放的数据块。该策略适用于特征图较大而卷积核较小的场景。而在广度优先策略中，每层卷积计算完毕后再进入下一层，卷积核使用后即可释放，因此更适用于特征图较小而卷积核较大的场景。

1. 指相同工艺下单位芯片面积能实现的计算能力更高。 [↑](#footnote-ref-1)
2. 本题中特指华为Davinci架构平台。 [↑](#footnote-ref-2)
3. 对一个有向无环图，将所有顶点排成一个线性序列，若对于图中任意一条边，在该序列中都出现在之前，则将该线性序列称为的一个拓扑序。 [↑](#footnote-ref-3)
4. 在调度序列上，一个数据缓冲区的生命周期定义为其ALLOC节点所在位置至FREE节点所在位置。 [↑](#footnote-ref-4)
5. 若前后两个数据缓冲区被分配至同一片物理缓存，则后一个缓冲区的ALLOC指令必须等前一个缓冲区的FREE指令执行结束后才能执行。 [↑](#footnote-ref-5)
6. 仅记录L1和UB类型缓存使用量。对于L0级缓冲区（L0A/B/C），其必须在申请后尽快释放。因此，在问题1给出的调度序列中，要求L0A、L0B、L0C上分别同时最多只能有一个缓冲区驻留。 [↑](#footnote-ref-6)
7. 请自行思考SPILL节点处理方式。 [↑](#footnote-ref-7)
8. 从单个执行单元的视角来看，它严格遵循调度序列中节点的排列顺序依次执行，不具备乱序执行能力。也就是说，如果节点b排在节点a之后，且两者属于同一执行单元，那么该单元必然先执行完节点a，才会开始执行节点b。 [↑](#footnote-ref-8)