基于 AI Engine 多核并行的二维卷积算法与系统部署

郑振东 程乾宇 蒋滨泽
zzd1411@mail.ustc.edu.cn qycheng@mail.ustc.edu.cn jiangbinze@mail.ustc.edu.cn
中国科学技术大学

版本：1.0

1 算法与系统设计

1.1 背景技术与硬件环境

二维卷积是一种针对矩阵数据的操作，在传统的图像处理与 AI 领域常用的卷积神经网络中被广泛使用。在二维卷积层中，卷积核是一个小尺度的二维权重矩阵，其工作机制是将卷积核在二维输入数据上“滑动”，并对当前输入的部分元素进行矩阵乘法，然后将结果汇为单个输出像素，组合为卷积结果。卷积核是对某个局部的加权求和，对应了对全局图像的局部感知，它的原理是在观察某个物体时我们既不能观察每个像素也不能一次观察整体，而是先从局部开始认识。

由于二维卷积算法的计算与访存特性，算法在通用处理器平台上不能获得很好的性能，以 GPU、FPGA 为代表的异构计算平台能够为这类算法带来更好的加速能效比。本设计使用 Xilinx VCK5000 Versal PCIe 加速卡所提供的硬件资源对二维卷积算法进行高效实现。

传统意义上，标量处理器（如 CPU）在具有不同决策树和广泛库的复杂算法中非常有效，但在性能扩展方面受到限制；矢量处理器（例如 DSP、GPU）在一组可并行计算函数集上效率更高，但受到存储结构的影响，存在较高的时延与较低的能效比。可编程逻辑（如 FPGA）可以精确地根据特定的计算功能定制，在实时应用和非规则数据处理方面表现最佳，但相关算法的更改会引入大量的编译时间。

VCK5000 提供了 ARM Cortex-A72、FPGA、DSP、AI Engine 等资源，同时对标量、矢量计算与定制计算进行支持，这些计算引擎可以通过高速的片上网络进行自定义互联。其中 AI Engine 是一种支持超长指令字（VLIW）、单指令多数据（SIMD）、多核处理的矢量处理器，作为硬部署的处理器核，用户可以直接为 AI Engine 编写具有数据级并行特性的内核程序，通过可编程逻辑（PL）侧，将数据从板卡 DDR4 内存直接传入 AI Engine 进行计算。AI Engine 与 FPGA 相比，不需要复杂的硬件考量，数据并行编程更加简单，适合于自适应 AI 推断应用。VCK5000 提供了一个包含 400 个 AI Engine 的阵列供用户使用。

1.2 主机-板卡交互平台

基于 X86-64 主机环境下的 VCK5000 板卡，本设计提供了可以批量运行 4K（3840*2160）分辨率图像二维卷积算法的平台。如**图 1**所示，具体执行流程如下:

1. 主机端将图像数据传送到板卡预留的 DDR4 内存输入缓冲区；
2. 在板卡的 PL 侧执行数据分发内核，将图像数据拆分为多个 64×32 的块，按序分发到若干个 AXI-Stream 接口向对应的 AI Engine 核心传递数据；
3. 板卡上的 AI Engine 阵列收到数据，执行 64×32 图像的二维卷积操作；
4. 在板卡的 PL 侧执行数据重组内核，将各个 AI Engine 处理小块的部分输出通过 AXI-Stream 接口取回，处理块间重叠数据重组为一张完整图像后，写入输出缓冲区；
5. 主机端将卷积结果数据从预留的 DDR4 内存输出缓冲区取回。

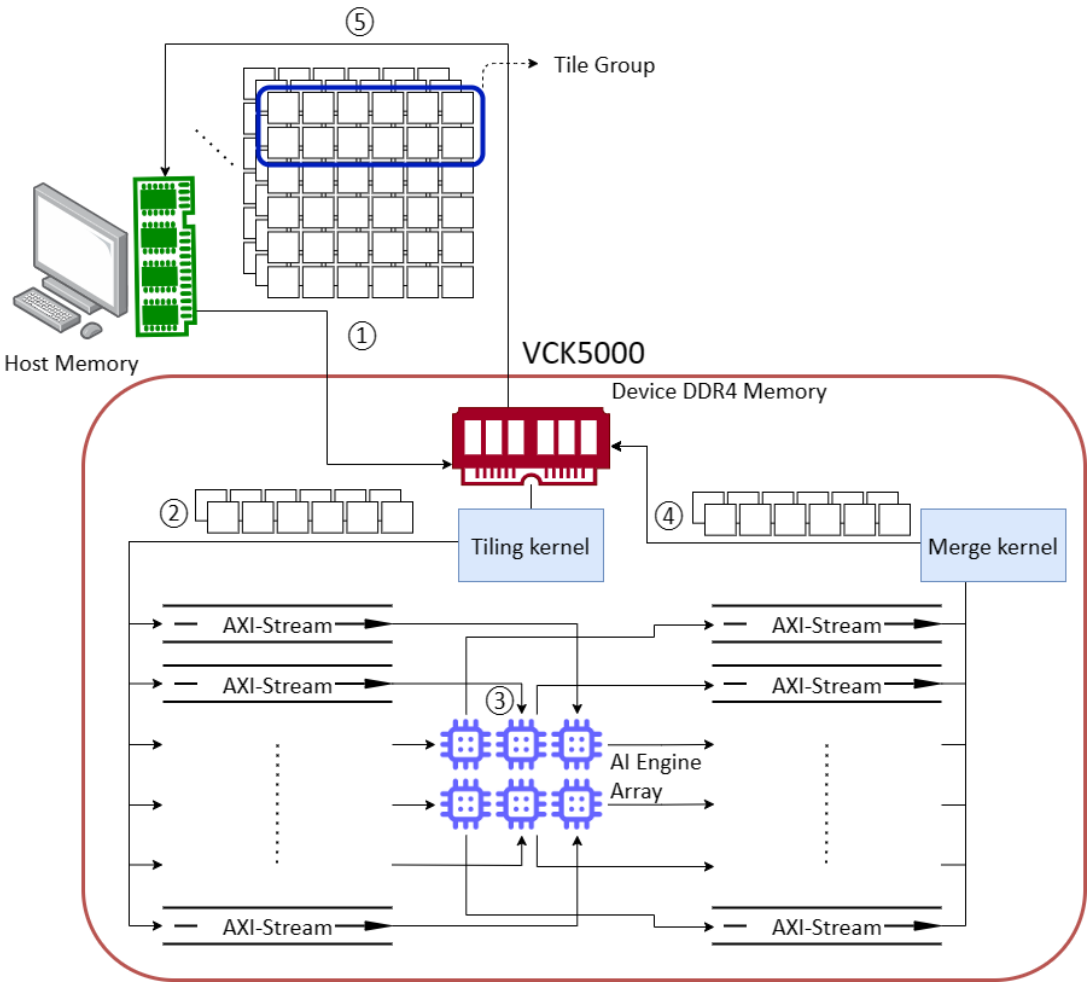

图 1: 平台整体框架

1.3 AI Engine 二维卷积内核

AI Engine 主要参照 Vitis Vision Library 中的 filter2d 实现进行修改，处理固定 64×32 大小的 int 类型矩阵卷积。处理时，预先对块数考虑 padding，按如下规则对数据进行复制后，再进行卷积：

1. padding 后矩阵四个角的数据与原矩阵四个角的数据相同；
2. padding 后矩阵第一列/行的数据与原矩阵第一列/行的数据相同；
3. padding 后矩阵最后一列/行的数据与原矩阵最后一列/行的数据相同。

由于 AI Engine 支持使用 intrinsic 进行 SIMD 编程，卷积内核中的计算语句可以一次实现多个数的乘累加操作，降低卷积操作的计算开销。

1.4 数据分发与重组

1.4.1 数据分发内核

为了使系统能够支持任意规模的图像输入，我们以 64×32 为分块大小，对由板卡 DDR4 内存传来的输入数据从上到下，从左到右进行分块处理。为了实现多个 AI Engine 的并行处理，我们以 12 个块为一组将分块按顺序组织起来，在一轮数据传输中，通过多个 AXI-Stream 通道将一组块数据中的每个块分别传输到对应的 AI Engine 执行卷积运算，并在计算完成后，将各 AI Engine 的执行结果重组。

由于二维卷积运算的数据复用特性，如果简单地将分块后两两之间毫无重叠的块数据传入并进行卷积，则无法通过各块的卷积输出组合获得完整的结果。为此，我们设计了一个输入分块机制更为复杂的

PL 端数据分发内核，使得每个传入 AI Engine 的块都与其输入数据上、下、左、右的分块有一个宽度为 2 的重叠。由此 PL 端可以通过 AI Engine 的输出结果还原出完整的卷积输出。完整图片的分块示意图如**图 2**所示。

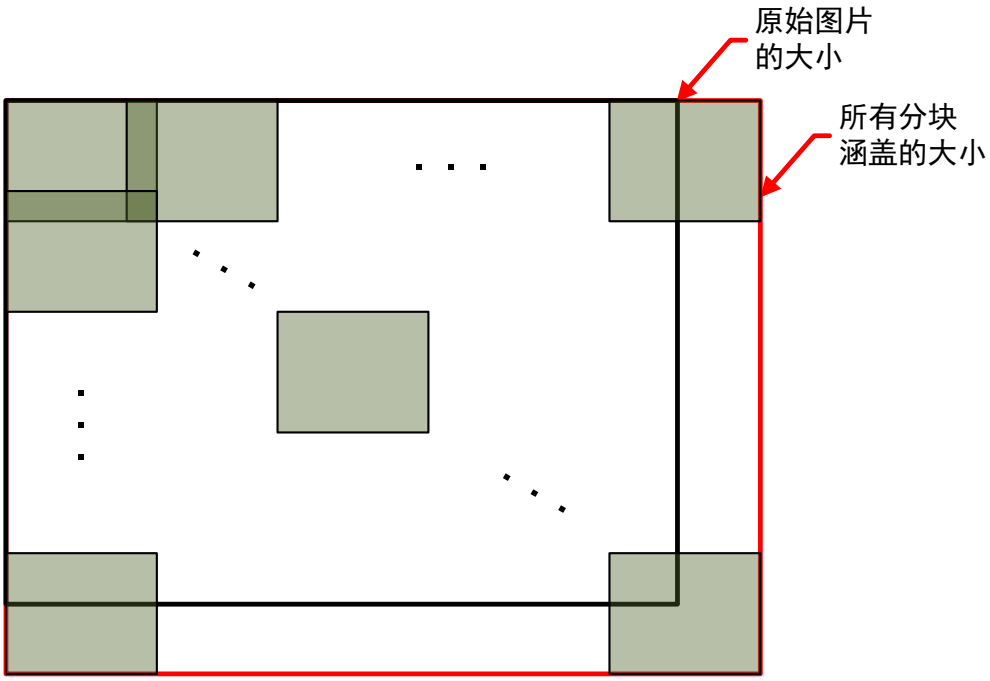

图 2: 数据分块布局

值得注意的是，对最后一行或最后一列的数据分块而言，由于预设的卷积 padding 规则，数据分发内核需要将该分块内的最后一行或最后一列有效的数据复制到下一行或下一列。除此之外，64×32 分块内没有有效数据的部分，全部用 0 进行填充。

1.4.2 数据重组内核

由于 AI Engine 会对接收的每个分块数据都预先 padding 再进行卷积，因此分块卷积的部分结果需要在重组时丢弃。为此，我们在 AI Engine 阵列后设计了专门的 PL 端数据重组内核，将各个分块的输出按照其在原始图片中的顺序进行组合，将位于左上角、左下角、右上角、右下角、第一行列、最后一行列的分块以及其他分块进行分类处理。同样的，重组内核执行过程中，一样通过 256 位数据位宽的内存输出接口传出数据。分类处理具体规则如下：

位于左上角的分块，其最后一列和最后一行的卷积结果是错误的（除了完整图片大小恰好为 64×32 等特殊情况）。如**图 3**(1) 所示，粉色区域代表 AIE 进行 padding 后得到的数据，红色区域代表错误的结果。重组内核直接将这个分块卷积结果都写入最终输出中，并由后续写入的分块输出对错误结果进行覆盖。按照从左至右，从上至下的顺序对分块结果进行拼接时，后续的分块会用正确的结果覆盖掉这些错误数据。

位于右上角的分块，其第一列与最后一行的卷积结果是错误的。除此之外，此分块右侧的一部分输入不属于原有效数据，如**图 3**(2) 所示，绿色区域数据是属于原有效数据的部分，蓝色区域数据是原有效数据最后一列的复制。重组内核直接将分块卷积结果中除第一列外未超过原图片区域的数据写入最终输出，因为后续的分块依旧会用正确的结果覆盖掉最后一行错误的数据。左下角、右下角的分块按照类似的规则处理。

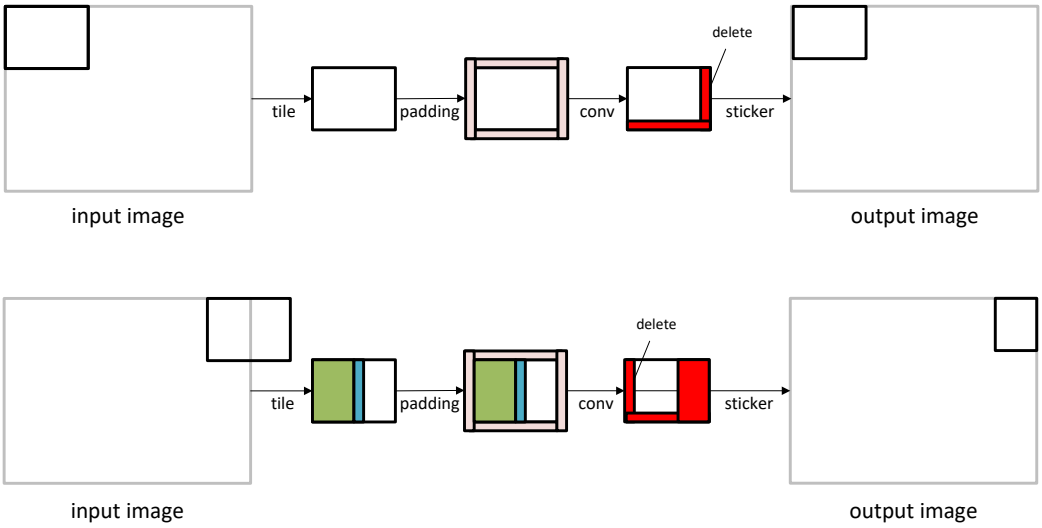

图 3: 无效数据处理

除了上述情况之外，位于第一列的分块，其第一行、最后一行、最后一列的卷积结果是错误的。与上述情况同理，重组内核直接将分块卷积结果中除第一行外的数据写入最终输出。第一行的分块按照类似的规则处理。位于最后一列的分块，其第一行、最后一行、第一列的卷积结果是错误的。与处在右上角的情况同理，此分块右侧的一部分输入不属于原有效数据。重组内核直接将分块卷积结果中除第一行、第一列外未超过原图片区域的数据写入最终输出。最后一行的分块按照类似的规则处理。

对其他的不处在边缘的分块而言，第一行列、最后一行列的卷积结果都是错误的。重组内核直接将分块卷积结果中除第一行、第一列外的数据写入最终输出。

2 实验分析与展望

实验使用 Xilinx 提供的异构加速计算集群（HACC），提供 16 核 Intel Xeon Gold 6246R X86-64 处理器，16GB 内存与支持 PCIe Gen4x8 协议、QDMA 的 VCK5000 板卡平台。

平台最终部署了 7 个 AI Engine 与相应的 AXI-Stream 输入输出通道，同时为板卡 DDR4 内存到 PL 侧内核部署了 256 位数据宽度的端口，有效利用了 DDR4 存储带宽。

AIE 编译结果如**图 5**所示，蓝色代表 AI Engine，紫色代表缓存。二维卷积实现合计消耗了 17 个 AIE 阵列中的块，消耗了 7 个块 AI Engine 计算，12 个块用作缓存，17 个块参与流式传输。

我们使用了随机生成的 32 位矩阵数据对平台进行测试，批量处理了 100 张图片。实验数据表明，平台处理单张 4K 分辨率数据的平均时间为 63ms，处理帧率为 15.87fps。其中数据传入传出的时延为 24.49ms，接近总时长的一半，说明计算开销没有影响到系统的整体性能，运行二维卷积算法的瓶颈主要在于访存。

目前的设计依旧存在诸多问题。首先，对目前的设计而言，从存储器端口取数据、向 AXI-Stream 端口发送数据两个任务之间是串行执行的，未来可以通过双缓存技术，使得存储器端口访存与 AXI-Stream 数据发送可以同时执行，覆盖一定的访存时延。其次，目前由于 QDMA 平台限制，二维卷积不能够有针对性地为连接到多个 PL 内核端口的 DDR4 缓存数据分配存储 bank，实现更为精细的内存并行传输，未来可以对平台设计进行进一步的优化。我们希望能够之后进一步迭代设计，最大化利用板卡带宽。

```
IMG 93/100 : Erro time: 0
IMG 94/100 : Erro time: 0
IMG 95/100 : Erro time: 0
IMG 96/100 : Erro time: 0
IMG 97/100 : Erro time: 0
IMG 98/100 : Erro time: 0
IMG 99/100 : Erro time: 0
IMG 100/100 : Erro time: 0
==========  END  ==========
********************************************************************************

       Average transefer time from host TO   device: 11.605167ms
       Average transefer time from host FROM device: 12.884778ms
       Average execution time                      : 63.006476ms

********************************************************************************
```

图 4: 运行结果

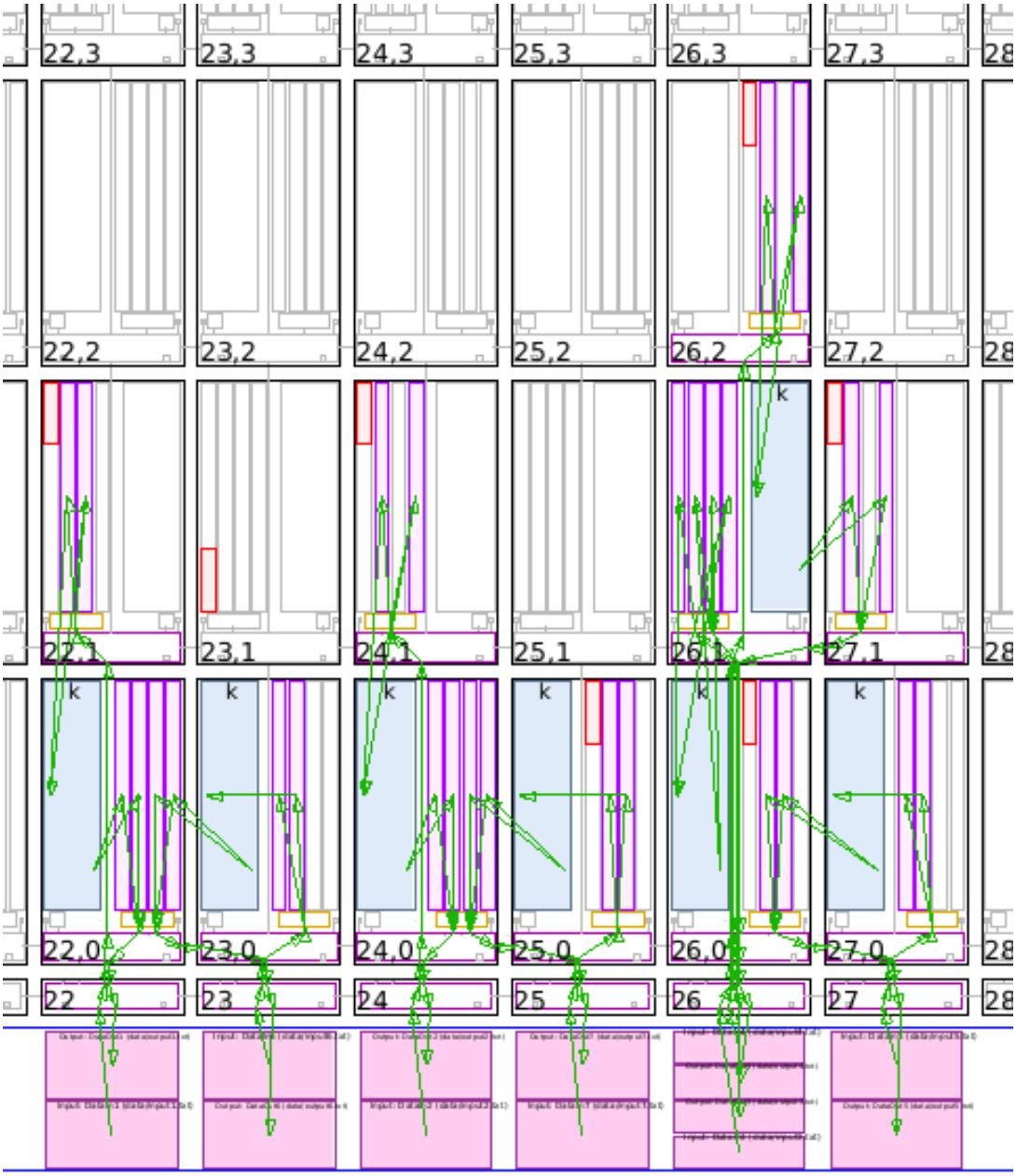

图 5: AI Engine 阵列部署布局