arXiv:2302.07594v1 [cs.DC] 15 Feb 2023

面向 Xilinx Versal 上的深度学习推理矩阵乘法

Jie Lei、José Flich、Enrique S. Quintana-Ortí系统与计算机信息学系 巴伦西亚理工大学 巴伦西亚，西班牙
{jlei,jflich,quintana}@disca.upv.es

摘要

深度神经网络对许多人工智能（AI）任务产生了显著的积极影响，推动了各种高性能算法以及专用处理器和加速器的发展。本文探讨了这一场景，并展示了在传统处理器架构中实现通用矩阵乘法（GEMM）所依据的原理，同样适用于在诸如嵌入 Xilinx Versal 平台的 AI Engine（AIE）瓦片这类特殊加速器上，实现深度学习（DL）运算的高性能。具体而言，我们在 Xilinx Versal VCK190 上对 GEMM 内核进行了原型实现，实验结果表明，对于 16 位整数操作数，其性能可达到 AIE 瓦片理论峰值的近 86.7%。

关键词 矩阵乘法 · 深度学习 · Xilinx Versal 人工智能引擎 (AIE) · 高性能

1 引言

在过去二十年中，摩尔定律逐渐放缓以及登纳德缩放定律的终结，促使人们采用多核处理器，随后又推动了领域专用加速器（例如 NVIDIA 张量核心和 Google TPU）以及非对称多核设计（例如 ARM big.LITTLE；Apple M1、M2；以及 Intel Alder Lake、Raptor Lake）的兴起 [1–4], 赛灵思并未忽视专用硬件在性能和功耗方面的优势，因此推出了 Versal AI Engine (AIE) Core。该设计包含一组计算引擎、高级 I/O 和集成式双倍数据速率控制器，旨在面向广泛的工作负载 [5, 6]。

对此，我们分析了如何将通用矩阵乘法（GEMM）映射到启用 *AIE* 的 Xilinx Versal 自适应计算加速平台（ACAP）上。我们之所以选择这一计算内核作为目标，主要是因为 GEMM 是许多科学和工程应用的基石。此外，在使用流行的卷积神经网络（卷积神经网络）和近年来兴起的 Transformer 进行深度学习（DL）训练和推理时，大多数算术运算都可以转换为 GEMM 的形式 [7, 8]。

针对在赛灵思 Versal 上高效实现 GEMM，我们作出以下两项主要贡献：

- 我们证明，在配备分层多级存储器和单指令多数据（SIMD）算术单元的传统处理器架构上实现 GEMM 的现代方法，同样适用于启用 AIE 的 Xilix Versal ACAP。为此，我们将矩阵操作数映射到存储层次结构的不同层级，并为 AIE 瓦片开发了架构专用的 微内核。

- 我们针对 Xilinx Versal VCK190 的具体情况，对算法的一般设计进行了定制。此外，我们还在该平台上进行了全面的实验分析，揭示了性能注意事项以及缓存配置参数所发挥的关键作用。

在此，我们强调，我们的设计面向 Versal VCK190，但将其适配到其他支持 AIE 的 Versal ACAP 上是非常直接的。

本文其余部分的结构安排如下：第 3 节简要介绍 Versal VCK190；第 4 节回顾通用矩阵乘法（GEMM）的现代实现。接着，第 5 节描述面向 Versal AIE 瓦片的 GEMM 微内核设计，以及如何将通用矩阵乘法算法映射到支持 AIE 的 ACAP 的存储器组织中。第 6 节对所提出的算法进行全面的实验评估；在第 **??** 节中，我们讨论研究所得的见解，并简要展望正在进行和未来的工作。最后，本文以文中使用的缩略语词汇表收尾。

2 相关工作

乘加（MAC）运算约占机器学习推理中算术运算的 80% [9]。这些 MAC 运算通常可以汇总到一个 GEMM 内核中，例如在卷积神经网络（CNN） [10]、Transformer [11], 和长短期记忆（LSTM）网络 [12] 中。因此，大量研究致力于加速 GEMM，方法是优化内存访问和/或借助硬件加速器提升内核速度。BLIS [13] 显著提高了存储器效率，并与面向卷积神经网络且适用于基于单指令多数据的加速器 [10, 15, 16]的近似计算 [14],相结合。本文将 BLIS 的设计映射到 Xilinx Versal 平台。

矩阵乘法是深度学习推理和训练中的基础内核。对于卷积神经网络（CNN）而言，IM2COL 变换 [8] 通过将输入激活张量转换为大得多的矩阵，把计算量巨大的卷积算子转化为通用矩阵乘法。对于许多卷积层，应用此变换后得到的通用矩阵乘法具有一个较大的维度和两个较小的维度。从这个角度来看，针对这些特定情况开发专用的通用矩阵乘法实现，往往能显著提升性能 [16]。

3 Xilinx Versal VCK190 ACAP 概述

3.1 异构架构

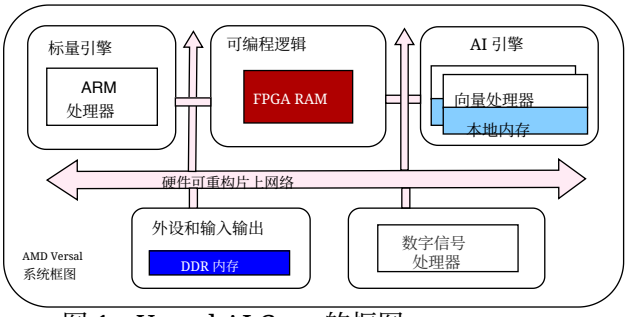

图 1：Versal AI Core 的框图。

Versal AI Core 由一系列异构 硅设计组成，旨在为边缘和云计算工作负载提供高性能；参见图 1。对于本研究所针对的特定 Versal VC1902 处理器 [5]，其架构包含：

1. 一颗双核 ARM Cortex-A72 处理器，用于计算密集型工作负载；另有一颗双核 ARM Cortex-R 5F 处理器，用于时间关键型任务。2. 1,968 个数字信号处理器，用于加速算术运算。
3. 一种可定制的现场可编程门阵列（或可编程逻辑），含有 899,840 个查找表。4. 400 个独立的“向量核心”，称为 AIE 瓦片，并排列成二维阵列。

除此之外，每个 AIE 瓦片都包含一个单指令多数据算术单元；这些向量核心共同构成了 VC1902 面向计算密集型深度学习工作负载的关键特性。具体而言，它们在 INT8 算术运算中每个时钟周期可执行多达 128 次乘加运算（每个 AIE 瓦片） [17]。此外，Versal VCK190 还包含可重新编程的片上网络，可在各模块之间实现高带宽通信。

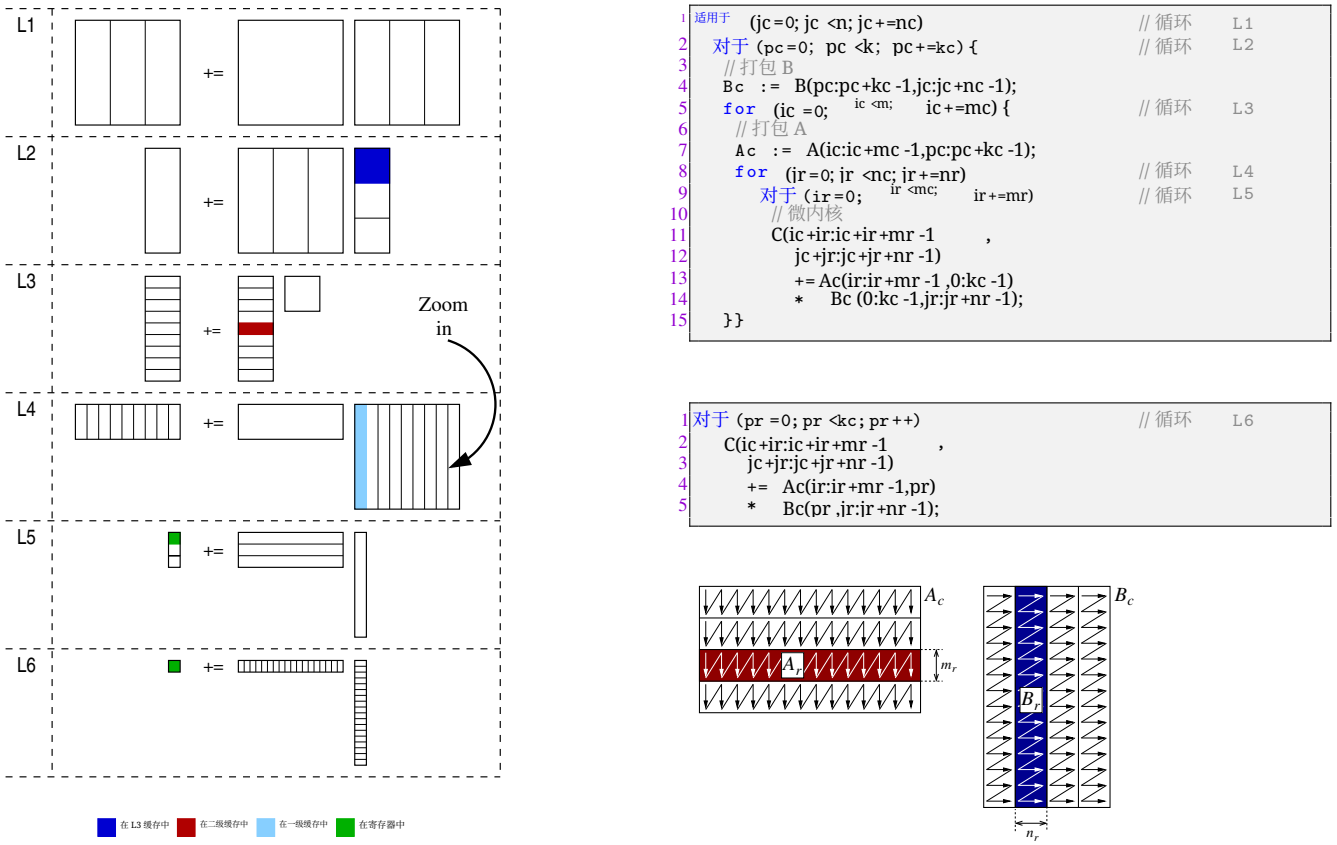

图 2：用于通用矩阵乘法的基线高性能算法。左侧：存储层次结构中的数据传输；右上：分块算法；右侧中部：微内核；右下：输入矩阵操作数的打包。

3.2 多功能存储架构

Versal VCK190 采用灵活的存储器架构，基本由以下部分组成：1）每个 AIE 瓦片配备 32 KB“本地”内存；2）分布式块随机存取存储器和超大容量随机存取存储器，容量分别为 4.3 MB 和 16.3 MB；3）2 GB 全局 DDR4 内存。此外，每个 AIE 瓦片都可以直接访问二维网格中“邻近”瓦片的 32 KB 内存，而其余 AIE 瓦片的本地内存则可通过直接存储器访问进行访问。专用互连与片上网络共同为 AIE 提供无阻塞、确定性的通信。

FPGA 的可重配置性使其能够利用块随机存取存储器和超大容量随机存取存储器作为 AIE 的临时缓冲区，数据通过流式接口直接馈送至 AIE 瓦片，以实现低延迟访问。AIE 瓦片通过片上网络访问全局 DDR4，以获得更高吞吐量。

4 高性能 通用矩阵乘法 与深度学习推理

4.1 通用矩阵乘法 在传统处理器架构中的应用

考虑 GEMM $C \mathrel{+}= AB$，其中矩阵 A 为 $m \times k$，矩阵 B 为 $k \times n$，矩阵 C 为 $m \times n$。现代的这一内核实现（例如 AMD AOCL、OpenBLAS、BLIS，以及可能还有 Intel oneAPI 中的实现）遵循 GotoBLAS2 [18] 的基本思想，通过围绕两个打包例程和一个依赖于架构的微内核构建五层嵌套循环，对矩阵操作数进行分块（切片） [19]；参见图 2 右上角，以及其中标记为 L1、L2、...、L5 的循环。

对于该基线算法，适当选择最外层三个循环的步长（也称为缓存配置参数） m_c、n_c、k_c，并谨慎地将矩阵输入安排到两个缓冲区中，A_c、B_c（其维度分别为 $m_c \times k_c$ 和 $k_c \times n_c$；见图 2 左侧和右上方），再通过打包

例程，大幅减少缓存未命中的次数 [20,21]。为简单起见，以下假设m、 n、 k分别是 m_c、 n_c、 k_c的整数倍。

此外，微内核还包含一个额外的循环（在图 2 右侧中部标记为 L6），它通过一系列秩一变换反复更新一个大小为 C 的微块，记作 C_r；每次变换都涉及一个 A_c 微面板的列和一个 B_c 微面板的行。这些微面板对应于图 2 右下方标记为 A_r 和 B_r 的 A_c,。在采用 SIMD 算术单元的当前处理器中，微内核的“维度”（由 $m_r \times n_r$ 给出）经过选择，以便在微内核循环内部实现向量化。为此，打包例程引入的专门数据布局可确保微内核以单位步长访问微面板中的数据，从而能够使用 SIMD 指令加载这些数据。

总之，为了实现高性能，这种通用矩阵乘法的表述依赖于三个参数（m_c, n_c, k_c），这些参数可以调整，以适配当前架构的分层存储系统（见图 2 左侧）；此外，还需要一个依赖于架构的微内核，例如可以对其进行调优以实现向量化。

4.2 通用矩阵乘法 与深度学习推理

近年来用于自然语言处理的 Transformer，其相当一部分算术运算开销来自形式为 $C \mathrel{+}= AB$ 的通用矩阵乘法，其中 A 是权重矩阵， B/C 是输入/输出激活矩阵。此外，对于广泛用于计算机视觉和通用信号处理任务的卷积神经网络，降维方法 [8] 会将卷积算子表示为大型通用矩阵乘法，其中 A 是滤波器矩阵， C 对应于输出激活矩阵，而 B 是对输入激活矩阵应用 IM2COL（或 IM2ROW）变换后得到的增广矩阵。为应对其高昂的存储开销，可以将这一变换分块，作为通用矩阵乘法实现 [16], 中打包过程的一部分，或者在直接卷积算法 [7] 中作为即时操作执行。

5 将通用矩阵乘法映射到 Xilinx Versal VCK190，以实现深度学习推理

5.1 在存储层次结构中分布数据

Versal VCK190 提供了多种可能性，可将问题数据分布到其存储资源中，以实现较高的数据可复用性。具体而言，由于 Versal VCK190 中不同存储层级具有不同的带宽速率和容量（见表 1），因此可根据目标算法的算术强度，利用这一特点，通过缓存友好型编程技术（例如分块算法 [19],），并将问题数据谨慎地分布到存储层次结构的各个层级，从而提高效率。
对于 Versal VCK190，一方面，DDR4 内存提供了充足的空间来容纳全局程序数据；在我们的问题中，这些数据对应于通用矩阵乘法的矩阵操作数 B, C 在深度学习推理场景下，矩阵操作数包含输入/输出激活值。另一方面，可编程逻辑中的随机存取存储器具有高存储吞吐量，可用作暂存区，临时存储权重/滤波器矩阵的完整预打包矩阵 A 。实际上，这意味着无需在全局内存中保留未打包矩阵 A 的副本。最后，快速且靠近处理器的本地内存可提供高吞吐量，可用于存储一个打包的微面板 B_r。这三个层级——全局内存、现场可编程门阵列 RAM 和本地内存——将分别发挥传统处理器中一级、二级和三级缓存的作用（见图 2 左侧），下文将对此进行讨论。

表 1：Versal VCK190 中的多级存储层次结构。

级别	容量
AIE 向量寄存器	2 KB
AIE 瓦片本地存储器	32 KB
现场可编程门阵列 RAM	20 MB
DDR4（全局）存储器	2 GB

图 3 以图示方式展示了在 Versal VCK190 上映射 GEMM 内核时所采用的矩阵操作数和打包缓冲区的分布。为简化起见，下文考虑一个 CNN 模型，以及对单个卷积算子应用 IM2ROW 变换后得到的通用矩阵乘法，同时考虑到通用矩阵乘法在深度学习推理场景中的特殊用法，以减少通信开销：

- 操作数 A 包含用于卷积的只读滤波器。因此，可以将整个矩阵预先打包（离线）到一组缓冲区 A_c 中，并存放在 FPGA RAM 中。这里我们利用了这样一个事实：深度学习模型部署后，会用于对大量输入样本进行推理。此外，对于 CNN 模型，

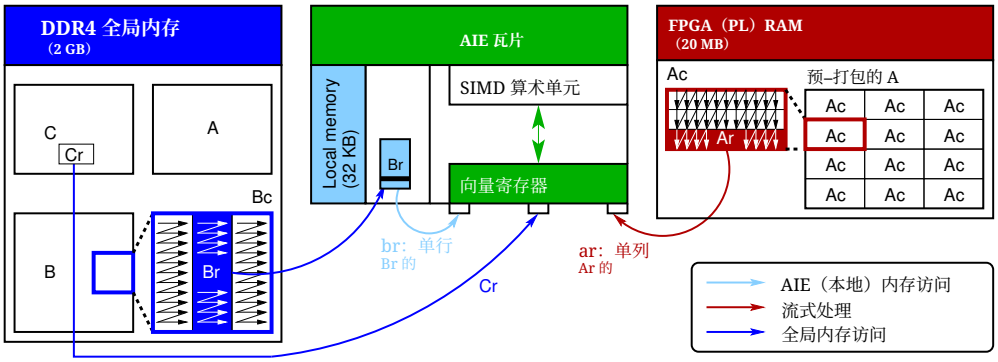

图 3：Versal VCK190 不同存储器之间的数据映射和传输。数据传输将全局内存中的微块 C_r 、FPGA 内存中的微面板 A_r 以及本地内存中的微面板 B_r 直接移至向量寄存器。全局内存中微面板 B_C 到本地内存中 B_r 的数据复制由一个标量引擎（ARM 处理器）执行。

FPGA RAM 通常足够大，可以容纳模型的所有滤波器参数（权重和偏置）。因此，我们可以预先打包 A 并将结果预加载到 FPGA 存储中，这样在推理过程中就不会产生任何开销。

- 矩阵 B 对应于卷积算子的激活输入。该操作数因推理样本而异，因此我们需要在通用矩阵乘法执行期间将其打包到缓冲区 B_c 中。遵循该操作的高性能实现方法，我们在 DDR4 全局内存中进行打包（因为 Versal ACAP 中没有与三级缓存等效的结构）。执行期间，各个微面板 B_r 也会通过 GMIO 复制到 AIE 本地内存。为此，AIE 瓦片通过片上网络直接访问全局内存。在这里，我们通过多次重用该块中的条目来摊销传输 B_r 的成本；参见图 2 右上方的循环 L5。
- 最后，矩阵 C 包含卷积算子的激活输出。每次执行微内核时，首先将一个维度为 $m_r \times n_r$ 的小型微块 C_r，从 DDR4 全局内存直接加载到 AIE 瓦片的向量寄存器中，并在微内核执行结束时写回；参见左侧和中右侧的图 2。

此时，值得指出我们的解决方案与在传统处理器上实现通用矩阵乘法时所采用的方法之间的一些差异：

- 从硬件角度来看，传统处理器集成了多个缓存层级（通常为两到三级），并依靠内存控制器来协调各层之间的数据移动。相比之下，在 Versal VCK190 中，传输操作需要明确编码到算法中。这给程序员增加了额外负担，但也能严格控制数据传输，因为中间存储器层的行为类似于暂存区。
- Versal 系统提供（窗口和）流式访问方法，用于与“外部世界”通信。流式处理可为 PL 到 AIE 的通信提供高吞吐量和更低延迟，但其本质上也不同于传统处理器的算术单元与缓存/存储器层级之间发生的通信。
- 从算法角度来看，鉴于目标是为深度学习推理实现通用矩阵乘法，我们可以预先将权重/滤波器矩阵 A 打包到一组存储在 FPGA RAM 中的缓冲区里。因此，无需像通用矩阵乘法基线算法那样，在循环 L3 中进行打包。

5.2 微内核的设计

每个 AIE 瓦片有四个累加器寄存器，可用于存储向量数据通路的结果。一个累加器寄存器的位宽为 768 位，可视为 16 个累加器通道，每个通道为 (768/16=) 48 位。为了获得高性能，累加器寄存器的位宽会影响微内核在实际应用中的维度。对于 Versal VC1902 上面向深度学习的通用矩阵乘法实现，我们采用 INT16 作为基准数据类型（稍后将在第 6.1 小节讨论），并将微内核的维度设为 $m_r \times n_r = 16 \times 4$；也就是说，由微内核的循环 L6 更新的微块 C_r 包含 4 列（累加器寄存器的数量）各 16 个元素（一个累加器寄存器的容量）。 C

```
# 定义      Bref(i,j) Br [i*nr+j]
void 微_内核 (输入_窗口_int16 *                    __restrict 双倍数据速率_输入,
                        输入_窗口_int16 *          __restrict 可编程逻辑_输入,
                        INT16 *Br  ,
                        输出_窗口_int16 *__restrict out) {

  // 向量       , ar 的累加器           , br , C
  v32int16 ar0, ar1;
  v16INT16 br;
  v16累加48 Cacc0  , Cacc1 , Cacc2 , Cacc3:

  // mac16 内在函数的参数
  无符号整数x 偏移量              = 0x73727170;
  unsigned int x 偏移量_hi  = 0x77767574;
  unsigned intx 平方             = 0x3120;
  unsigned intz 偏移量           = 0x0;
  unsigned int z 偏移量_hi   = 0x0;

  for (无符号整数 i=0;       i<Kc/4;   i+=1)
    // 循环展开，以重叠传输和计算
    chess 准备 用于 流水线处理
    chess_循环_范围 (Kc/4,)          {
    // 将一个包含 16 个 INT16 的向量读入 br
    br = *(v16int16*) Bref[i];

    // 读入 ar
    ar0 = upd_w(ar0   ,0, window_readincr_v16(可编程逻辑_IN));
    ar0 = upd_w(ar0   ,1, window_readincr_v16(可编程逻辑_IN));
    ar1 = upd_w(ar1   ,0, window_readincr_v16(可编程逻辑_IN));
    ar1 = 更新_w(ar1   ,1, window_readincr_v16(可编程逻辑_IN));

    // 执行两次 mac16()
    Cacc0 = mac16(Cacc0,  ar0 ,0, x 偏移量,   x 偏移量_hi,     x 平方,   br ,0 , z 偏移量,   z 偏移量_hi, 4);
    Cacc1 = mac16(Cacc1,  ar0 ,0, x 偏移量,   x 偏移量_hi,     x 平方,   br ,1 , z 偏移量,   z 偏移量_hi, 4);
    Cacc2 = mac16(Cacc2,  ar0 ,0, x 偏移量,   x 偏移量_hi,     x 平方,   br ,2 , z 偏移量,   z 偏移量_hi, 4);
    Cacc3 = mac16(Cacc3,  ar0 ,0, x 偏移量,   x 偏移量_hi,     x 平方,   br ,3 , z 偏移量,   z 偏移量_hi, 4);
    //
    Cacc0 = mac16(Cacc0,  ar1 ,0, x 偏移量,   x 偏移量_hi,     x 平方,   br ,8 , z 偏移量,   z 偏移量_hi,4);
    Cacc1 = mac16(Cacc1,  ar1 ,0, x 偏移量,   x 偏移量_hi,     x 平方,   br ,9 , z 偏移量,   z 偏移量_hi,4);
    Cacc2 = mac16(Cacc2,  ar1 ,0, x 偏移量,   x 偏移量_hi,     x 平方,   br ,10, z 偏移量,   z 偏移量_hi, 4);
    Cacc3 = mac16(Cacc3,  ar1 ,0, x 偏移量,   x 偏移量_hi,     x 平方,   br ,11, z 偏移量,   z 偏移量_hi, 4);
  }

  // 从全局内存读取 Cr
  v16int16 C0 = 窗口_递增读取_v16（双倍数据速率_输入）;
  v16int16 C1 = 窗口_递增读取_v16（双倍数据速率_输入）;
  v16int16 C2 = 窗口_递增读取_v16(双倍数据速率_IN);
  v16int16 C3 = 窗口_递增读取_v16(双倍数据速率_IN);

  // 转换结果，将其加到 Cr 并写回存储器
  C0=运算符 +(srs(Cacc0,0),C0); 窗口_递增写入(out,C0);
  C1=运算符 +(srs(Cacc1,0),C1); 窗口_递增写入(out,C1);
  C2=运算符 +(srs(Cacc2,0),C2); 窗口_递增写入(out,C2);
  C3=运算符 +(srs(Cacc3,0),C3); 窗口_writeincr(out,C3);
}
```

图4：AIE 瓦片微内核的简化版本。

图 4 展示了我们为 VC1902 编写的微内核。经过一些初始声明后，代码包含一个循环（第 19 行），对应 L6，该循环遍历微面板 A_r，B_r 的 kc 维度。每次迭代中，循环体将 A_r 的一列元素（16 个元素，位于 ar0 或 ar1 中）与 B_r 的一行元素（4 个元素，位于 br 中，通过宏 Bref 访问）相乘，并将中间结果累加到四个不同的寄存器上（第 27–49 行的乘加操作）。

人工智能引擎（AIE）的内置函数 mac16() 可在一个周期内执行 32 次 INT16 乘加运算（MAC）。对于 INT16 数据类型，每次乘加操作可涉及一个最多包含 32 个元素的向量，以及另一个最多包含 2 个元素的向量。然而，我们的微内核（$m_r \times n_r = 16 \times 4$）与该内置函数的输入维度不匹配。因此，为了使维度 $n_r = 4$ 满足 mac16() 的要求，我们将 16×4 乘加操作分为两部分；参见第 33 行和第 42 行。这样，额外调用一次 mac16() 内置函数可将输入量额外增加四倍，从而充分利用该函数。循环体的每次迭代都会从 FPGA 内存中读取四行 A_r（64 个元素，见第 27–30 行），并从本地内存中读取 B_r 的 4 列（16 个元素，见第 24 行）。因此，每次循环迭代都会从靠近算术单元的存储层级中读取 64+16 个元素，

并对其执行 $64 \cdot 16 \cdot 2 = 2{,}048$ 次算术运算。这样有助于通过足够的算术计算来摊销内存传输的成本，原则上可实现计算受限微内核。
将计算拆分为两部分可实现较高的计算通信比。此外，累加器和向量寄存器的高利用率（分别占总资源的 100% 和 75%），结合编译器优化参数，有助于将乘加操作与数据传输重叠，从而提升性能。

循环完成后，代码会将这些中间结果“传输”到矩阵 C。为此，微内核的每次执行都会从全局内存加载一个 16×4 微块 C_r（第 53–56 行），更新其内容后，再将结果存回全局内存（第 59–63 行）。如果 k_c 足够大，则这些最终数据传输的开销可以摊销（后文将对此进行讨论）。

6 实验结果

本节中的通用矩阵乘法算法评估使用 Xilinx Vitis 2022.1 开发工具完成。采用 AIE 事务级 System C 模拟器对设计的时序、资源需求和汇编指令进行性能分析，从而实现准确的性能分析 [22]。

6.1 不同数据类型的近似计算

精度缩放是一种重要的近似计算技术，常用于边缘设备上的量化深度学习推理，以降低能耗，同时保持结果具有可接受的精度。针对这一点，Versal VC1902 支持多种数据类型的算术运算，可借此提升性价比。具体而言，Versal 提供多种实际的数据精度格式。AIE 支持 INT8、INT16 和 FLOAT32，但不同格式的性能差异显著，精度越低，所需的 AIE 周期越少。例如，在一个时钟周期内，acAIE 可执行 128 次 UINT8 乘加运算；INT16 为 32 次，FP32 为 8 次。

这些数据促使我们选择 INT16 作为面向 Versal VCK190 的面向深度学习的通用矩阵乘法内核的基准数据类型，因为该数据类型能够提供适当的精度，同时减少存储器占用并加快计算。此外，针对 INT16 数据类型的 mac16() 内置函数支持灵活的实现。未来的工作中，我们将调整微内核的通用设计，以适配更低精度。

6.2 AIE 瓦片微内核的评估

在第 6.2 小节中，我们说明了选择 $m_r \times n_r = 16 \times 4$ 微内核（用于 INT16 数据类型）的理由。虽然也有其他微内核尺寸能够产生正确结果，但选择较小的微内核会导致：1）使用较少的累加器寄存器/通道，这可能会使累加器算术单元出现空闲周期，从而降低性能；2）循环 L6 主体的算术内存比更差。此外，若微内核超出寄存器/通道的最大数量，编译器可以透明地处理：在微内核执行期间，将某些数据临时保存到存储器中（并最终恢复）。遗憾的是，如果这种寄存器溢出发生在循环 L6 内部，性能可能会显著下降。

为说明这一点，我们进行了以下两项实验。第一项实验中，我们对使用 k_c = 256 的 16×4微内核执行情况进行了性能分析。此配置下，微内核内部执行的乘加运算共 16,384 次，设计耗时 596 个周期完成，相当于每周期 27.5 次乘加运算。AIE 瓦片使用 INT16 并在累加器寄存器的 16 个通道上进行累加时，理论峰值为每周期 32 次乘加运算。此处性能略有下降，是因为需要从全局存储器加载微块 C_r、对其进行更新，并将结果写回存储器。

在第二种配置中，我们采用了 32×4 微内核，理论上，当该微内核驻留在本地内存中时，其 C_r 重用率更高。然而，这种方法超出了累加器寄存器的容量，导致寄存器溢出。具体而言，使用 k_c = 256 时，该设计执行的乘加运算次数是前一种设计的两倍，因此理论开销应大致相当于 $596 \cdot 2 = 1$，192 个周期。但在实践中，完成计算需要 1,429 个周期，比预期长约 20%。因此，第二种配置每周期仅能实现 23 次乘加运算，远未达到 AIE 的峰值。

我们以最后两点说明结束前面的讨论：

- Versal VC1902 中的累加器寄存器可视为 4 个 768 位累加器。我们的微内核包含 8 个 mac16() 内在函数，这些函数使用这 4 个相互独立的累加器寄存器，因此将此类资源的利用率提升到了 100%。在保持 INT16 数据类型的同时增大微内核的规模

可能会提高寄存器空间的利用率，但最终会导致寄存器溢出。当前设计使用了向量寄存器资源中 75% 的空间。实际上，本节介绍的第二种场景（使用 32×4 微内核）超出了向量寄存器资源的最大容量。因此，在微内核循环执行期间，其中一些额外的向量需要暂时存储在本地内存中，之后再取回向量寄存器，这会导致我们希望避免的性能下降。 [17]

• 采用 48 位累加器的目的是获得 16 位乘法结果，并对这些结果进行累加而不发生位溢出。在 VCK190 中，48 位是累加器的最小“尺寸”。

6.3 选择缓存配置参数 k_c

确定微内核的尺寸后，接下来我们将研究如何为 k_c 选择最佳值。请注意，我们沿着层次结构，从处理器寄存器逐步转向速度较慢的存储器层级。因此，在上一小节中，我们结合累加器寄存器的数量讨论了 $m_r \times n_r$ 的取值；在本小节中，我们将考虑本地存储器的容量来选择 k_c ；在下一小节中，我们将研究如何根据 FPGA RAM 的容量设置 m_c 的最佳值。

理想情况下，我们希望 k_c 尽可能大，因为从微内核的角度来看，与算术运算次数 $(2\, m_r \cdot n_r \cdot k_c)$ 相比，这种选择可以减少将微块 C_r 从全局内存加载到内存或保存回全局内存所产生的开销（该层级上的 $m_r \times n_r$ 次读取，以及同等次数的写入）。此外，驻留在本地存储器中的微面板 B_r 为 $k_c \times n_r$，较大的 k_c 意味着对该层级资源的重用效果更好。然而，在 $n_r = 4$ 已经确定的情况下， k_c 的最大尺寸受本地存储器容量限制。此外，由于 k_c 是通用矩阵乘法的 k 维分块参数，从应用角度来看，我们受到约束：$k_c \leq k$。

在本小节中，我们首先展示 k_c 对微内核性能的影响。为此，我们分析了微内核单独执行时，乘加运算（有用算术运算）所占周期数与总周期数（包括加载/存储数据产生的开销）之比。图5展示了

该参数对性能的影响。例如，在 $k_c = 64$ 的情况下，微内核执行总共需要 212 个周期，其中 150 个周期对应乘加运算，其余周期则用于数据传输。因此，该配置的效率约为 60%。随着 k_c 增大，其逐渐接近水平渐近线：当 $k_c = 128$ 分别为 256 和 290 时，效率分别为 75.3%、85.9% 和 87.6%。当 $n_r = 4$ 时，最多为 $k_c = 290$，以确保微面板 B_r 能够容纳在本地内存中。

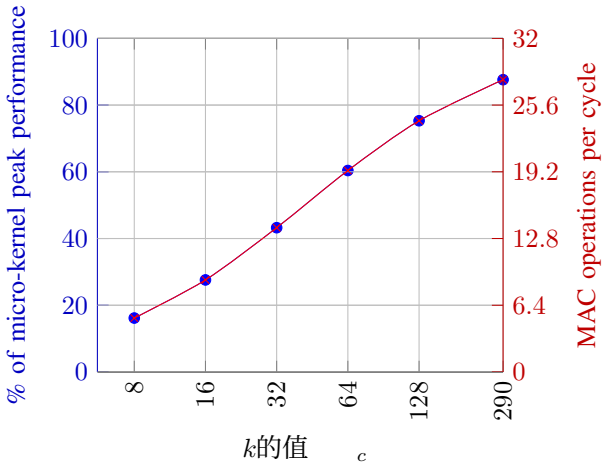

图5： k_c 对 $m_r \times n_r = 16 \times 4$ 微内核性能的影响。

该实验表明，微内核的算术运算与数据传输之比很大程度上取决于缓存配置参数 k_c 的取值。特别是，通过选择合适的（较大） k_c 值，可以基本隐藏将 C_r 从全局内存复制到本地内存所带来的开销。此外，选择合适的微内核维度，有助于将从本地内存复制 B_r 以及从 FPGA 内存复制微面板 A_c 所造成的负面影响降低到约 12% 的周期。

这里需要特别强调的是，一旦固定了 k_c，这就为 GEMM 内核的性能设定了上限，因为完整算法是以微内核作为基石构建而成的。唯一的

我们尚未考虑的遗漏数据传输，因为它们不发生在微内核内部，对应于将全局内存中的缓冲区 B_c 复制到本地内存中的微面板 B_r；参见图 3。这些复制由 VCK190 中的标量引擎（ARM 处理器）负责调度，其开销将在下一小节中讨论。

6.4 选择缓存配置参数 m_c

现在考虑驻留在 FPGA RAM 中的缓冲区 A_c，其维度为 $m_c \times k_c$，并采用与分析 k_c 时类似的推理方法。首先，m_c 有两个上界：1）根据上一小节设定的 k_c，可以很容易地从 FPGA 内存容量推导出 m_c 的最大维度；2）从应用角度来看，由于 m_c 是由问题维度 m 决定的分块参数，因此我们有 $m_c \leq m$。接下来，我们通过分析将数据从全局内存中的缓冲区 B_c 复制到 AIE 瓦片本地存储器中的微面板 B_r 所产生的开销，来讨论选择较大 m_c 值的好处。为此，举例来说，假设 $k_c = 290$ 且 $n_r = 4$。在这种配置下，全局内存与本地存储器之间的数据复制会带来显著开销：复制耗时 8,309 个周期，而执行单个微内核耗时 8,952 个周期。因此，每周期的乘加运算数会降至仅 2。幸运的是，这种数据复制每次循环 L4 迭代只发生一次，复制的微面板 B_r 随后会在 m_c/m_r 次微内核执行中重复使用（每次循环 L5 迭代执行一次）。因此，在 m_r 已经固定的情况下，这一推理表明应选择较大的 m_c 值，因为较大的比率 m_c/m_r 有助于通过微内核内部足够多的算术运算来摊销复制 B_r 的成本。

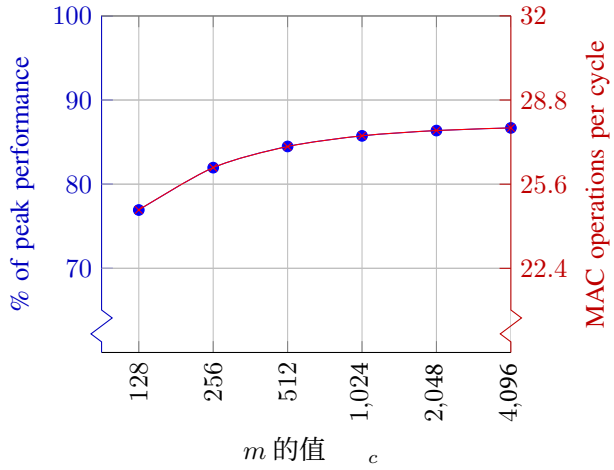

图 6：m_c 对 GEMM 内核性能的影响。在此实验中，$m_r \times n_r = 16 \times 4$，$(m, n, k)$ =(4,096, 4,096, 290)，$n_c = n$和 $k_c = k$。

本节最后，我们探讨了 m_c 对特定通用矩阵乘法问题性能的实验影响。为此，我们选择了(m, n, k) =(4,096, 4,096, 290)，设置 $n_c = n$，$k_c = k$，并在多次执行 GEMM 内核时改变了 m_c。图 6 展示了选择较大的 m_c所带来的性能提升，但同时也揭示出性能明显趋于水平渐近线，约为峰值性能的 87.6%（对应每周期 32 次 INT16 乘累加运算，或 64 次 INT16 算术运算）。因此，这与单独考察微内核（即 GEMM 内核的构建模块）时的性能完全吻合；图5报告的结果显示，其性能最高可达峰值性能的 86.7%。这意味着，只要将 m_c 设得足够大，就可以利用算术运算隐藏 B_c和 B_r 之间复制所带来的开销。

作为对前述分析的粗略总结，我们实现的 Versal 通用矩阵乘法（GEMM）因将数据从 A_c 和 B_r 复制到向量寄存器而损失了 15% 的峰值性能，此外，由于 B_c 和 B_r之间的数据传输，又损失了 4–5%。

7 术语表

ACAP：自适应计算加速平台；**AIE**：人工智能引擎；**API**：应用程序编程接口；**CNN**：卷积神经网络；**DDR**：双倍数据速率（存储器）；**DL**：深度学习；**DSP**：数字信号处理器；**GMIO**：全局存储器输入/输出；**INT8**：8 位整数（算术运算）；**LUT**：查找表；**MAC**：乘加运算；**NoC**：片上网络；**PL**：可编程逻辑；**SIMD**：单指令多数据；**TOPS**：每秒万亿次运算；**TPU**：张量处理单元；GEMM：通用矩阵乘法；**RAM**：随机存取存储器。

致谢

作者谨此感谢欧盟地平线2020研究与创新计划通过玛丽·斯克沃多夫斯卡-居里资助协议（编号：956090）提供的资助（APROPOS，http://www.apropos-itn.eu/）。
本研究还获得了西班牙科学与创新部/西班牙国家研究署（MCIN/AEI/10.13039/501100011033）研究项目 PID2020-113656RB-C22 的资助，以及欧洲区域发展基金的资助 建设欧洲的一种方式。此外，欧洲高性能计算联合承办机构（联合承办机构）通过资助协议（编号：955558）为本研究提供了资助（eFlows4HPC 项目）。该联合承办机构获得欧盟地平线2020研究与创新计划以及西班牙、德国、法国、意大利、波兰、瑞士和挪威的支持。

参考文献

[1] G. E. Moore，《在集成电路上集成更多元件——转载自《电子学》杂志第38卷第8期，1965年4月19日，第114页及以后》。*IEEE*固态电路学会通讯，第11卷第3期，第33–35页，2006年。[2] R. Dennard、F. Gaensslen、H.-N. Yu、V. Rideout、E. Bassous和A. LeBlanc，《采用极小物理尺寸离子注入MOSFET的设计》，*IEEE*固态电路期刊，第9卷第5期，第256–268页，1974年。[3] J. L. Hennessy和D. A. Patterson，《计算机体系结构的新黄金时代》，*Commun. ACM*，第62卷第2期，第48–60页，2019年。[4] N. P. Jouppi 等，《数据中心张量处理单元的性能分析》，*SIGARCH*计算机体系结构新闻，第45卷第2期，第1–12页，2017年6月。[在线]。可访问：https://doi.org/10.1145/3140659.3080246[5] S. Ahmad 等，《赛灵思首款7纳米器件：Versal AI Core（VC1902）》，载于*2019*年第*31*届*IEEE Hot Chips*研讨会（*HCS*），2019年，第1–28页。[6] M. Voogel 等，《Xilinx Versal Premium》，载于*2020*年第*32*届*IEEE Hot Chips*研讨会（*HCS*），2020年，第1–46页。[7] S. Barrachina 等，《面向ARM处理器上高性能深度学习推理的直接卷积重构》，《系统架构期刊》，2022年，即将刊出。[8] K. Chellapilla、S. Puri和P. Simard，《用于文档处理的高性能卷积神经网络》，载于手写识别前沿国际研讨会，2006年，INRIA报告inria-00112631，网址：https://hal.inria.fr/inria-001126。[9] A. Castelló、S. Barrachina、M. F. Dolz、E. S. Quintana-Ortí、P. S. Juan和A. E. Tomás，《利用通用优化技术和BLIS在多核ARM处理器上实现高性能、高能效的深度学习推理》，《系统架构期刊》，第125卷，文章编号102459，2022年。[在线]。可访问：https://www.sciencedirect.com/science/article/pii/S1383762122000509[10] C. Ramírez、A. Castelló和E. S. Quintana-Ortí，《面向基于RISC-V指令集架构的GAP8处理器的类BLIS机器学习矩阵乘法》，《超级计算期刊》，第78卷第16期，第18051–18060页，2022年5月28日。[11] H. Peng、S. Huang、T. Geng、A. Li、W. Jiang、H. Liu、S. Wang和C. Ding，《利用列平衡块剪枝在FPGA上加速基于Transformer的深度学习模型》，载于*2021*年第*22*届国际电子设计质量研讨会（*ISQED*），2021年，第142–148页。[12] W. Zhang、F. Ge、C. Cui、Y. Yang、F. Zhou和N. Wu，《基于FPGA的LSTM加速器设计与实现》，载于*2020*年第*20*届*IEEE*通信技术国际会议（*ICCT*），2020年，第1675–1679页。[13] F. G. Van Zee和R. A. van de Geijn，《BLIS：快速实例化BLAS功能的框架》，*ACM* 数学软件汇刊，第41卷第3期，2015年6月。[在线]。可访问：https://doi.org/10.1145/2764454

[14] F. G. Van Zee、D. N. Parikh和R. A. V. D. Geijn，《在BLIS框架中支持混合域混合精度矩阵乘法》，*ACM* 数学软件汇刊，第47卷第2期，2021年4月。 [在线]。可访问：https://doi.org/10.1145/3402225[15] A. Castelló、E. S. Quintana-Ortí和F. D. Igual，《BLIS矩阵乘法算法系列剖析》，载于*2022*年第*30*届*Euromicro*并行、分布式与网络处理国际会议（*PDP*），2022年，第92–99页。[16] P. San Juan、A. Castelló、M. F. Dolz、P. Alonso-Jordá和E. S. Quintana-Ortí，《面向多核处理器的高性能、可移植卷积算子》，载于*2020*年第*32*届*IEEE*计算机体系结构与高性能计算国际研讨会（*SBAC-PAD*），2020年，第91–98页。[17]《AI Engine 内核编码最佳实践指南（UG1079）》，https://docs.xilinx.com/r/en-US/ug1079-ai-engine-kernel-coding，访问日期：2022年5月。[18] K. Goto和R. A. van de Geijn，《高性能矩阵乘法剖析》，*ACM* 数学软件汇刊，第34卷第3期，第12:1–12:25页，2008年5月。[19] K. Dowd和C. R. Severance，《高性能计算》，第2版。O’Reilly，1998年。[20] F. G. Van Zee和R. A. van de Geijn，《BLIS：快速实例化BLAS功能的框架》，*ACM* 数学软件汇刊，第41卷第3期，第14:1–14:33页，2015年。[21] T. M. Low 等，《高性能BLIS的解析建模已足够》，*ACM* 数学软件汇刊，第43卷第2期，第12:1–12:18页，2016年8月。[22] 赛灵思，《AI Engine 工具与流程用户指南（UG1079）》，https://docs.xilinx.com/r/en-US/ug1079-ai-engine- kernel-coding/Tools，2022年10月，访问日期：2022年10月。