CHARM：基于 Versal ACAP 架构组合异构加速器实现矩阵乘法

Jinming Zhuang匹兹堡大学 美国
jinming.zhuang@pitt.edu

Jason Lau加利福尼亚大学洛杉矶分校，美国 l
au@cs.ucla.edu

Hanchen Ye伊利诺伊大学厄巴纳-香槟分校，美国
hanchen8@illinois.edu

Zhuoping Yang匹兹堡大学 美国
zhuoping.yang@pitt.edu

Yubo Du匹兹堡大学 美国
yubo.du@pitt.edu

Jack Lo
超威半导体公司，美国
jack.lo@amd.com

Kristof Denolf
超威半导体公司，美国
kristof.denolf@amd.com

Stephen Neuendorffer超威半导体公司，美国
stephen.neuendorffer@amd.com

Alex Jones
匹兹堡大学 美国
akjones@pitt.edu

Jingtong Hu
匹兹堡大学 美国
jthu@pitt.edu

Deming Chen
伊利诺伊大学厄巴纳-香槟分校，美国
dchen@illinois.edu

Jason Cong加利福尼亚大学洛杉矶分校，美国
cong@cs.ucla.edu

Peipei Zhou
匹兹堡大学 美国
peipei.zhou@pitt.edu

arXiv:2301.02359v1 [cs.AR] 6 Jan 2023

摘要

稠密矩阵乘法（MM）是深度学习应用中使用最广泛的内核之一。为了应对这些应用的高计算需求，同时采用FPGA和专用集成电路加速器的异构架构已成为很有前景的平台。例如，AMD/Xilinx Versal ACAP架构将通用CPU核心和可编程逻辑与针对人工智能/机器学习优化的AI Engine处理器相结合。由400个以1 GHz运行的AI Engine处理器组成的阵列，针对32位浮点（单精度浮点）数据，性能最高可达6.4 TFLOPs。然而，机器学习模型通常同时包含大型和小型矩阵乘法操作。大型矩阵乘法操作可以在多个核心之间高效并行化，而小型矩阵乘法操作通常无法实现这一点。我们观察到，在Versal ACAP上使用大型单体矩阵乘法加速器执行BERT自然语言处理模型中的一些小型矩阵乘法层时，其性能不到理论峰值性能的5%。因此，一个关键问题随之产生：对于包含多个不同规模矩阵乘法层的端到端应用，在通信带宽受限的情况下，我们如何设计加速器以充分利用丰富的计算资源？

我们发现，单体加速器拥有海量计算资源，而应用中的各个矩阵乘法层规模较小，两者不匹配由此造成了最大的系统吞吐量瓶颈。为解决这一问题，我们提出 CHARM 框架，将**多个不同架构的矩阵乘法加速器**组合起来，使其在单个应用中并行处理不同层。CHARM 包含解析模型，可指导设计空间探索，以确定加速器分区和层调度。为简化系统设计，CHARM 可自动生成代码，从而支持全面的板上设计验证。我们将 CHARM 框架部署在四种不同的深度学习应用程序上，包括 BERT、ViT、NCF 和多层感知机，并使用 AMD/Xilinx Versal ACAP VCK190评估板进行测试。实验结果表明，对于 BERT、ViT、NCF 和多层感知机，我们分别实现了 1.46 TFLOPs、1.61 TFLOPs、1.74 TFLOPs 和 2.94 TFLOPs 的推理吞吐量，相较于单体加速器，吞吐量分别提升了 5.29×、32.51×、1.00×和 1.00×。

计算机分类系统

• 计算机系统组织 → 异构（混合）系统；• 硬件 → 软硬件协同设计。

关键词

异构架构，领域专用加速器，Versal ACAP，映射框架，矩阵乘法，深度学习

ACM参考格式：

Jinming Zhuang、Jason Lau、Hanchen Ye、Zhuoping Yang、Yubo Du、Jack Lo、Kristof Denolf、Stephen Neuendorffer、Alex Jones、Jingtong Hu、Deming Chen、Jason Cong、Peipei Zhou。2023年。《CHARM：面向 Versal ACAP 架构的矩阵乘法异构加速器组合》。载于*2023*年*ACM/SIGDA*国际现场可编程门阵列研讨会论文集（*FPGA '23*），*2023*年*2*月*12*日至*14*日，美国加利福尼亚州蒙特雷。ACM，纽约，纽约州，美国，12页。https://doi.org/10.1145/3543622.3573210

1 引言

稠密矩阵乘法（MM）是许多深度学习工作负载中使用最广泛的内核之一，包括用于自然语言处理的 BERT [1] 、用于推荐的 NCF [2]、用于视觉分类的 ViT [3]，以及用于多层感知机分类或回归的 MLP [4]。根据谷歌 [5], 的分析结果，2017年，稠密矩阵乘法任务占谷歌数据中心神经网络（NN）推理工作负载的90%。这些应用程序日益复杂，对计算和数据移动提出了极高的需求。

根据 [6, 7, 8, 9],，片外带宽一直是影响系统性能和能效的瓶颈，而当前平台的一个普遍趋势是，片外带宽的扩展速度赶不上计算资源的扩展速度。因此，

FPGA '23，*2023*年*2*月*12*日至*14*日，美国加利福尼亚州蒙特雷© 2023年版权归所有者/作者所有。本文为作者版本，仅供个人使用，特此发布。请勿转载。最终的正式版本发表于*2023*年*ACM/SIGDA*国际现场可编程门阵列研讨会论文集（*FPGA '23*），*2023*年*2*月*12*日至*14*日，美国加利福尼亚州蒙特雷，https://doi.org/10.1145/3543622.3573210。

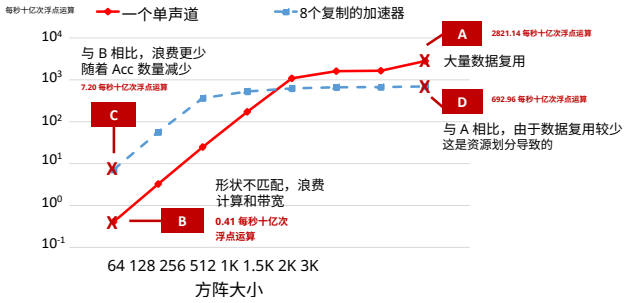

图 1：不同规模下方阵矩阵乘法的吞吐量。

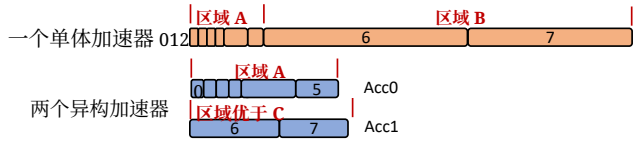

图 2：VCK190 上 BERT 的单体式矩阵乘法设计与多样化矩阵乘法加速器设计的执行时间线。

第一个研究问题由此产生：如何利用扩展速度较慢的片外带宽，维持扩展速度较快的计算能力？

一种常见的解决方案是在加速器（Acc）中分配更多片上存储，以提高数据复用率。如 [9]，中的渐近分析所示，矩阵乘法的片外通信总量按 $O(\frac{1}{\sqrt{M}})$ 的规模增长，其中 M 是片上分块大小。增大分块大小可以减少通信总量，从而减轻片外带宽的压力。

在本研究中，我们针对 AMD/Xilinx Versal ACAP 架构 [10]，，该架构将通用 CPU 核心和可编程逻辑（PL）与针对人工智能/机器学习计算优化的 AIEngine 处理器（AIE）相结合。例如，我们在 AMD/Xilinx VCK190 开发板上使用 384 个 AIE 和超过 80% 的片上 URAM 和 BRAM 资源实现了一个矩阵乘法加速器。图 1 中的红线展示了该加速器的性能。该设计以 1536×128×1024 为原生分块大小，在执行大型方阵矩阵乘法的分块计算时，可实现 2.8 TFLOPs 的吞吐量（点 A）。然而，若将不同规模的矩阵乘法直接映射到此类设计上，当方阵矩阵乘法尺寸低于 512 时，性能会显著下降，因为每个数据块都要填充到加速器的原生分块大小。例如，在点 B，此类单体设计的性能降至 0.41 GFLOPs，比点 A 低 6880×。尽管填充是利用大型加速器实现小规模矩阵乘法操作的一种常见且简单的方法，但填充会浪费计算资源和带宽。

填充的一种替代方案是实现多个具有更小原生数据块大小的加速器，并可能在每个加速器上并行执行不同任务 [11]。我们采用这种方法，使用八个彼此独立、原生数据块大小为 256×128×256 的加速器，如图 1 中的蓝色虚线所示。对于大小为 64 的小型方阵矩阵乘法操作，该方法在点 C 达到 7.2 每秒十亿次浮点运算，相较于点 B 的加速比约为 17×。

然而，较小的加速器规模也意味着大型矩阵乘法的数据复用较少，当操作规模大于 256 时，总吞吐量几乎达到饱和。当矩阵乘法规模为 3072（点 D）时，八个同构加速器的总吞吐量比单体设计中的点 A 小 4.08×。

这些实验揭示了两个相互冲突的设计目标。首先，我们希望实现大型矩阵乘法操作，并通过充分的数据复用，在设备上达到尽可能高的性能。其次，我们希望实现小型矩阵乘法操作，同时尽量降低计算和通信开销。上述两种简单设计似乎都无法同时实现这些设计目标。因此，第二个研究问题随之而来：对于大型和小型矩阵乘法层共存的真实端到端应用，如何在这两个设计目标之间进行权衡？

为了说明这些相互冲突的设计目标如何影响实际机器学习模型的性能，我们选取 BERT [1] 作为具有大小不同的矩阵乘法层的代表性工作负载。在 BERT 的一个 Transformer 层中，共有 8 种矩阵乘法内核，其中内核 0-5 是大型矩阵乘法，内核 6 和 7 是批量点积，即小型矩阵乘法。详细形状参见表 5。以内核 5 和内核 6 为例，内核 5 是形状为 3072×1024×4096 的矩阵乘法，内核 6 是形状为 96×512×512×64 的批量点积，这意味着共有 96 个大小为 512×512×64 的小型独立矩阵乘法。

如图 2 所示，使用一个单体矩阵乘法加速器时，内核 0-5 消耗了 BERT 矩阵乘法计算操作总量的 92%，但仅占矩阵乘法加速器总时间的 12%。相比之下，内核 6-7 消耗了总操作量的 8%，却占用了矩阵乘法加速器总时间的 88%。内核 0-5 位于区域 A（其表现与图 1 中的点 A 类似），该区域内加速器的吞吐量超过每秒 2082 亿次浮点运算。内核 6-7 则位于区域 B，该区域内加速器的吞吐量仅为每秒 236 亿次浮点运算。由于时间线上有很大一部分加速器执行资源未得到充分利用，矩阵乘法加速器的整体吞吐量仅为每秒 276 亿次浮点运算。我们能否为 BERT 设计一种方案，使其既位于区域 A（适用于大型矩阵乘法），又处于优于点 C 的区域（适用于小型矩阵乘法，且计算/带宽浪费较少或没有浪费）？

我们的答案是“是”。核心思想是，如图 2 所示，在双异构加速器系统中，为专门计算较大矩阵乘法的加速器分配更多资源，为其他加速器分配较少资源，使其同时计算较小的矩阵乘法。为实现我们的设计目标，我们需要应对这些新挑战。**首先**，我们需要提高每个加速器的计算利用率，即使用较小的加速器减少小型矩阵乘法中的资源浪费，并使用较大的加速器最大限度地提高大型矩阵乘法的数据复用率。**其次**，为最大限度地提高整体利用率，同时保持高吞吐量和低延迟，我们需要通过协同优化工作负载和资源划分，谨慎地重叠这些加速器的执行时间。**第三**，为便于开展设计空间探索（DSE），我们需要解析模型，以在资源和带宽约束下优化整体吞吐量。**第四**，为减少系统实现所需的编程工作，我们需要自动代码生成。**第五**，为解决运行多个加速器时应用程序图中各内核之间的依赖关系，我们需要一种加速器运行时，用于将不同任务中的内核调度到各个加速器上。

为回答这些研究问题，我们提出了 CHARM 架构及其配套的自动化框架——CHARM 框架。我们的贡献总结如下：

• **Versal 上的 CHARM 系统化设计方法**：为提高每个加速器的计算与通信效率，我们在第 4 节提出了针对 Versal 异构平台的全面设计方法。我们进一步提供了自动化 CHARM 设计空间探索（CDSE），以寻找最优的单个加速器配置。据我们所知，这是首项对 Versal 上系统化的数据移动与计算进行详细分析的工作。

• **CHARM 架构与框架：** 为了实现设计目标，即在应用程序中兼顾小规模和大规模矩阵乘法的良好性能，我们在第 5 节提出了 CHARM 架构和 CHARM 框架，以寻找优化后的设计。CHARM 框架包含多个模块。首先，在 CDSE 的基础上，我们提出 CHARM 多样化加速器组合器（CDAC），其采用基于排序的两步搜索算法，以多项式时间复杂度而非指数时间复杂度寻找优化后的 CHARM 设计。此外，为实现系统实施自动化，我们提出 CHARM 自动代码生成（CACG），用于生成 AIE、可编程逻辑和主机 CPU 的源代码文件。最后，在主机 CPU 上启动 CHARM 运行时系统（CRTS），将不同内核调度至各加速器，以优化任务延迟和整体系统吞吐量。

• 我们在第 6 节中将 CHARM 框架部署到 VCK190 上，以加速四种应用程序。板载实验表明，CHARM 对 BERT、ViT、NCF 和多层感知机的推理吞吐量分别达到 1.46 TFLOPs、1.61 TFLOPs、1.74 TFLOPs 和 2.94 TFLOPs；与一个单体加速器相比，吞吐量分别提升了 5.29×、32.51×、1.00× 和 1.00×。

• **面向 Versal 的白盒开源工具**。 AMD 为用户提供了一种用于神经网络应用的黑盒 IP，称为 DPU [11],；而我们将工具完全开源为白盒，并提供详尽的分步指南，帮助复现本文展示的所有结果，也便于其他用户学习并将其用于端到端系统。（***https://github.com/arc-research-lab/CHARM***）

2 相关工作

为了实现高吞吐量和高能效，神经网络加速器通常采用大量处理单元（PE），并共享类似的存储层次结构。也就是说，大部分数据存储在片外存储器中，同时还设有多个层级的片上缓冲区，包括连接到各个 PE 的本地存储器和全局共享内存，以进一步减少与片外存储器之间代价高昂的数据移动。许多研究通过探讨数据复用机会、计算并行性以及数据流的选择，为神经网络加速器的发展作出了贡献。

然而，许多既有工作采用一刀切的单体设计，无法高效处理形状和大小差异极大的层（Eyeriss [12, 13], ShiDiannao [14], NPU [15, 16, 17] 等 [18, 19, 20, 21]）。AutoSA [22] 是一种基于多面体的编译框架，可为稠密矩阵生成单体脉动阵列设计。Sextans 和 Serpens [23, 24] 是面向稀疏矩阵的通用单体加速器。 [25, 26] 分析布局和流水线效率。AMD DPU [11], 、Mocha [27] 等其他工作则通过在设备上分配多个重复的加速器来探索任务级并行性，但并未

表1：与先前工作的比较。

先前工作	One Mono	多重复	多种多样	工作负载分配	专用化 -tion for 加速器
Eyeriss 等 [12]-[26]	✓	×	×	×	×
DPU 等 [11, 27]	✓	✓	×	×	×
深度神经网络 Expl. 等 [28, 29]	✓	✓	✓	×	×
Herald [32]	✓	✓	✓	✓	×
CHARM（我们的方法）	✓	✓	✓	✓	✓

针对每个加速器进行专门化设计。DNNBuilder [28] 根据每层的操作数量，为每一层设计专用加速器。DNNExplorer [29] 通过将前几层的专用加速器与其余层的单体加速器相结合，对 DNNBuilder 进行了改进。虽然它采用了多个加速器，但缺乏对工作负载分配的全面探索。TETRIS [30] 和 TANGRAM [31] 提出了神经网络层内部及层间的多种数据流优化，以提升性能和能效。尽管它们提供了多样化的加速器设计，却缺少为实现整体高吞吐量而进行的设计空间探索和工作负载分配。Herald [32]提出了一种包含多个异构加速器的架构，并探索工作负载分配和资源划分。然而，它从候选池中选取若干现有加速器设计，例如 ShiDiannao [14], 和 NVDLA [33]，而没有对每个加速器进行设计空间探索。FPCA [34] 和 CHARM' 12 [35] 提出了完全流水化且可动态组合的粗粒度可重构架构，并通过置换网络，为应用程序中的不同内核组合松耦合加速器，但这会占用大量芯片面积。

总之，我们在表1中总结了我们的方法与以往工作的区别。我们的方法能够从单体、多个重复加速器和多个不同加速器中选择设计；每个加速器都是专门设计的，会考虑我们的设计空间探索所涵盖的不同工作负载分配、数据流和数据并行策略。

3 VERSAL ACAP 架构概述

本节首先在第3.1节介绍 AMD/Xilinx Versal ACAP 架构的系统架构，然后在第3.2节介绍 AIE 阵列的存储器模型。

3.1 Versal ACAP 架构

图 3 展示了 VCK190 [36] 开发板的整体架构，并突出显示了顶部的 AIE 阵列。VCK190 开发板具有：（1）第一代 AIE 架构，其中包含 8×50**1 GHz** 七路超长指令字处理器，支持最高 1024 位的向量运算 [37], ；（2）ARM 处理器，用于运行 Linux 和通用应用程序；（3）可编程逻辑，用于设计专用硬件，并可集成数字信号处理器（DSP）。AI 引擎内核和 ARM CPU 可使用 C/C++ 代码编程，而可编程逻辑则可通过寄存器传输级和 C/C++ 代码编程，后者使用高级综合（HLS） [38, 39, 40, 41, 42, 43]。这三

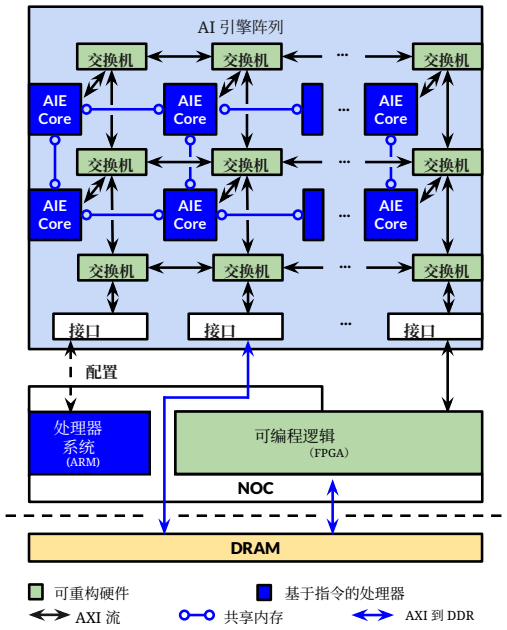

图 3：Versal ACAP 架构。

个组件与 PCIe 和 DRAM 控制器等输入/输出外设集成，并通过片上网络（NoC）构成异构片上系统。VCK190 开发板配备一条 DDR4-DIMM 片外存储器，峰值带宽为 25.6 GB/s。

3.2 AIE 内存模型

每个 AIE 处理器 tile 包含 32 KB 数据，并能够与四个方向上相邻的 AIE 共享数据（AIE↔相邻 AIE）。除了与相邻数据块共享本地存储器外，每个 AIE 数据块还连接到 AXI 流（AXIS）交换网络，从而支持 AIE 处理器之间的非本地通信（AIE↔非本地），并通过 39 个接口 tile 中的 PLIO 与 PL 通信（PL↔AIE）。VCK190 开发板提供 PL 与 AIE 之间 1.2 TB/s（PL↔AIE）/0.9 TB/s（AIE↔PL）的带宽，该带宽 46× 高于 DDR4 与 PL 之间的带宽。AXIS 交换机支持端口之间的电路交换连接和分组交换连接。电路交换连接提供专用且确定性的通信，并支持广播，即将来自单个输入通道的数据同时传输到多个输出通道。分组交换连接允许根据每个数据包起始处的目的地标头，将输入通道中的数据动态路由到不同目的地。这使得数据流可以在单一路由路径上进行时分复用。可以使用分组交换连接的一种情形是 AIE 的计算与通信（CTC）比率大于 1。在 AIE 0 进行计算时，分配给该 AIE 的端口处于空闲状态，因此可以通过分配一个与 AIE 1 目的地 ID 相匹配的不同标头，将数据传输到另一个 AIE，例如 AIE 1。

4 CHARM 单加速器设计

在本节中，我们将在第 4.1 节介绍如何使用数百个 AIE 为单个矩阵乘法加速器设计数据流与映射策略。然后，在

```
// 片外 <-> 片上时间循环
循环(整数i.0=0;i.0<TX;i.0++) //TX=M/(TI*A*X)
循环（整数j.0=0;j.0<TZ;j.0++) //TZ=N/(TJ*C*Z)
对于(整数k.0=0;k.0<TY;k.0++) //TY=K/(TK*B*Y)
  copyDataFromOffChipOnChip(...)
  // PL 片上缓冲区复用时间循环
  对于(整数 i.1=0;i.1<X;i.1++) //X
  适用于(整数 j.1=0;j.1<Z;j.1++) //Z
  对于(整数 k.1=0;k.1<Y;k.1++) //Y
    copyDataFromOnChiptoAIE(...)
    // AIE 阵列空间循环
    循环(整数 i.2=0;i.2<A;i.2++) //A
    循环(整数 j.2=0;j.2<C;j.2++) //C
    循环(整数 k.2=0;k.2<B;k.2++) //B
        // 单个 AIE 二维 SIMD 向量化循环
        for(int i.3=0;i.3<TI;i.3++)
        对于(整数 j.3=0;j.3<TJ;j.3++)
        对于(整数 k.3=0;k.3<TK;k.3++)
            ...
            二维单指令多数据(i.3,j.3,k.3);
```

清单1：矩阵乘法循环分块和数据流的伪代码。

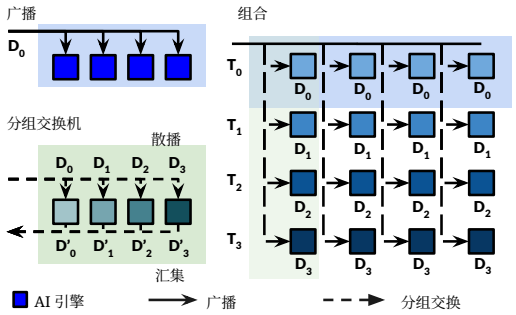

图4：结合广播电路交换连接和数据包交换连接，以减少 AIE 阵列所需的输入/输出。

第 4.2 节中，我们将介绍数据复用优化，以平衡庞大的计算并行性，以及 AIE 之间和 PL↔AIE 与 PL↔DDR 之间的通信。

4.1 单个矩阵乘法加速器的数据流与映射策略

清单1展示了基本稠密矩阵乘法的整体四级分块和映射策略。最内层循环分块（第16-20行）在单个 AIE核上实现矩阵乘法，并通过发出完全流水化的二维单指令多数据（向量-矩阵乘法）指令，充分利用指令级并行和数据级并行。每个 AIE 存储一个 (TI×TK) 左侧矩阵和一个 (TK×TJ) 右侧矩阵，并在其本地存储器中计算一个 (TI×TJ) 输出矩阵。第二内层循环数据块（第12-14行）表示在 AIE 阵列中的不同 AIE 核心之间进行执行空间分布。这些循环完全展开，并由 (A×B×C) 个 AIE 核心并行计算。空间分布也对应于所需输入/输出的数量，我们将在第4.2节讨论这一点。第三内层时间循环分块（第7-9行）表示对存储在可编程逻辑片上存储器中的数据进行顺序处理。来自片上可编程逻辑缓冲区的数据被馈送至 AIE 阵列 (X×Y×Z) 次，而来自 AIE 阵列的中间部分和则在可编程逻辑上累加。最外层循环（第2-4行）表示对存储在片外存储器中的数据进行时间处理，从而能够处理无法装入片上存储器的大型矩阵。循环边界可由整体输入矩阵大小 (M, K, N) 决定。

4.2 多个层级中的数据复用

设计每个加速器时，我们采用自底向上策略，并探索各层级的数据复用。首先，在单个 AIE 层级，我们充分利用 AIE 向量处理器的七路超长指令字能力，通过复用 AIE 本地寄存器和本地存储器，实现完全流水化的乘加运算。

其次，在 PL↔AIE 层级，当向数十或数百个 AIE 传输数据时，由于连接 AIE 阵列与 PL 的 PLIO 数量远少于 AIE 核心总数，我们通过利用数据广播和分组交换机制（见第3.2节）来减少所需的 PLIO 数量。图 4 展示了如何结合广播与分组交换，复用单个 PLIO 端口。假设有一个 4×4 AIE 阵列，用于计算大小为 1×4×4 的矩阵乘法（每个 AIE 执行 1 次乘加运算），每个 AIE 需要一个周期来获取左侧操作数（LHS）和右侧操作数（RHS），并需要四个周期完成一次乘法，因此计算与通信比为 4。借助矩阵乘法中的数据复用机会（例如，LHS 的一行可供 RHS 的不同列复用），我们可以如实线所示，在时间 0 利用一个 PLIO 端口将 LHS 的第一个数据广播到 AIE 阵列的第一行。在时间 1，通过指定不同的目的地标头，我们可以复用同一个 PLIO 端口，将 LHS 的第二个数据传输到 AIE 阵列的第二行。在时间 2 和时间 3，LHS 的第三个和第四个数据分别发送到 AIE 的第三行和第四行。在时间 4，AIE 的第一行完成计算，PLIO 也完成了向 AIE 第四行的数据传输。因此，在这种情况下，我们可以使用一个 PLIO 端口向 16 个 AIE 发送 LHS 数据，而不会造成任何性能下降。

第三，在 PL↔DDR 中，我们进一步为每个 acc 分配三组片上缓冲区，分别用于存储左侧、右侧和输出矩阵，以便大小为 (X×A×TI) × (Y×B×TK) 的左侧数据块能够在片上复用 (Z×TJ) 次。右侧矩阵和输出矩阵的缓冲区大小及复用率也可采用相同方法计算。此外，双缓冲技术应用于这三个缓冲区，以便将片外数据传输与计算重叠。通过充分挖掘多级数据复用机会，我们的系统能够在有限的片外带宽下保持较高的计算效率，即 VCK190 上 DIMM-DDR4 提供的 25.6 GB/s 带宽。

5 CHARM 架构与 CHARM 框架：组合多种不同的加速器

本节将在第 5.1 节介绍 CHARM 架构，并在第 5.2 节概述 CHARM 框架。随后，我们将在第 5.3 节至第 5.6 节讨论框架中的各个模块。

5.1 CHARM 架构

图 5 展示了 CHARM 架构：系统中包含一个或多个不同的矩阵乘法加速器，以及用于端到端深度学习应用中非矩阵乘法内核的其他内核加速器。我们将 AIE 阵列划分为多个矩阵乘法加速器（本例中为两个）。对于每个矩阵乘法加速器，我们根据分块策略设计一个专用的 DMA模块，其中包含数据传输控制逻辑和片上缓冲区。不同的 AIE分区通过 PLIO 接口和片上网络与各自对应的 DMA模块通信。我们将 AIE 阵列、其对应的 PLIO、

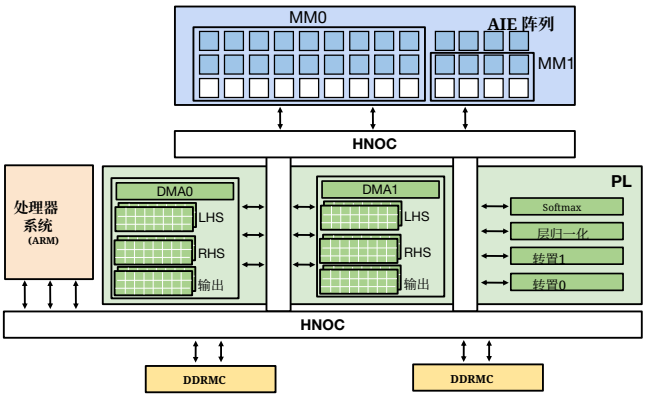

图 5：多个多样化矩阵乘法加速器及其他非矩阵乘法加速器的系统架构。

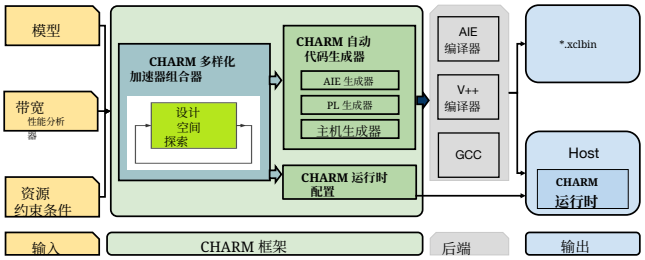

图 6：CHARM框架概述。

以及 DMA模块统称为一个矩阵乘法加速器设计。对于每种非矩阵乘法内核，例如 BERT、ViT 模型中的转置、Softmax 和层归一化，我们在可编程逻辑侧为每种内核设计一个加速器。每个非矩阵乘法加速器都包含 DMA、计算逻辑和本地缓冲区。对于这些通信受限内核，设计目标是达到接近峰值的片外带宽。运行这些内核时，由于它们会占用全部片外带宽，因此我们选择在矩阵乘法加速器之前或之后，依次启动这些非矩阵乘法内核和通信受限内核。

5.2 CHARM框架概述

我们在图 6 中展示了所提出的 CHARM 框架概述。CHARM 框架以应用模型、平台片外带宽分析结果和平台硬件资源约束为输入，执行自动化优化和代码生成，并调用后端编译器生成可直接运行的二进制文件作为输出。CHARM 框架包含几个模块：(1) CDAC 基于 CDSE，寻找吞吐量最高的最优设计，并输出每个加速器的可配置设计参数。此外，它还会生成运行时配置文件，指定特定内核应调用哪个加速器。(2) CACG 以 CDAC 生成的可配置参数为输入，实现设计，并生成 AIE、可编程逻辑和主机 CPU 所需的全部源代码文件。CHARM 调用相应的后端工具，生成硬件位流和主机二进制文件。(3) CRTS 以 CDAC 提供的运行时配置文件和内核依赖图为输入，将任务池中的内核调度到可用的加速器上。

5.3 面向单加速器的 CHARM 设计空间探索（CDSE）

设计空间探索 可配置参数：A、B、C、X、Y、Z。为了使各种加速器都能达到优化后的吞吐量，我们设计了 CDSE。它以矩阵尺寸（M、K 和 N）、可选的用户指定硬件约束，以及硬件平台片外特征数据库为输入，并执行基于解析模型的搜索。在 CDSE 过程中，我们将单个 AIE 工作负载设为 32×32×32，即 TI=TK=TJ=32。在该设计点，我们使矩阵乘法的内核效率最高达到 95%，利用了 75% 的 AIE 本地存储器，并获得 4 的计算与通信比。CDSE 的输出是满足所有硬件约束的可配置参数，包括 **A、B、C、X、Y、Z**。参数 A、B、C 决定 AIE 阵列中所用 AIE 和 PLIO 的数量。X、Y、Z、A、B、C 与预设参数 **TX、TY、TZ**共同决定所用片上缓冲区的数量。该优化问题可表述为如下所示的整数规划（IP）优化问题。$AIE_{\text{数量}}$、$PLIO_{\text{输入}}$、$PLIO_{\text{输出}}$ 和 片_上随机存取存储器表示用户指定的硬件约束：

$$\max\ Throughput = M \cdot K \cdot N \cdot 2/TIME \tag{1}$$

$$\text{s.t.} A \times B \times C \le AIE_{num} \tag{2}$$

$$\begin{aligned} Port_{in} &\le PLIO_{in}, \\ Port_{out} &\le PLIO_{out} \end{aligned} \tag{3}$$

$$Buff \le On_chip_{RAM} \tag{4}$$

AIE 阵列分块选择。 由于 {A, B, C} 已完全展开并映射到 AIE 阵列，因此展开因子 A、B 和 C 的乘积应小于或等于公式 2 中 AIE 的总数。分组交换端口的数量由 {A, B, C} 以及第 4.2 节所述的 I/O 复用机制决定。它们应满足输入和输出 PLIO 资源约束。输入和输出 PLIO 的数量可通过以下公式计算：

$$\begin{aligned} Port_{in} &= \lceil A \cdot B/CTC \rceil \\ &+ \lceil C \cdot B/CTC \rceil \\ Port_{out} &= \lceil A \cdot C/CTC \rceil \end{aligned} \tag{5}$$

PL 分块选择。 分配片上 PL 缓冲区是为了通过提高数据复用率，弥补片外到 PL 与 PL 到 AIE 阵列之间 46 倍的带宽差距。公式 6 展示了左侧、右侧和输出缓冲区的大小，以及片外到片上的通信时间。BP 指每个数据的字节数，BW_L,R,O 是通过带宽分析测得的片外带宽。

$$\begin{aligned} Buff_L &= (X \cdot A \cdot TI) \cdot (Y \cdot B \cdot TK) \cdot BPD \\ Buff_R &= Y \cdot Z \cdot B \cdot C \cdot TK \cdot TJ \cdot BPD \\ Buff_O &= X \cdot Z \cdot A \cdot C \cdot TI \cdot TJ \cdot BPD \\ Buff &= 2 \cdot (Buff_L + Buff_R + Buff_O) \\ Time_{L,R,O} &= Buff_{L,R,O}/BW_{L,R,O} \end{aligned} \tag{6}$$

性能建模。为了计算总执行时间，需要考虑片外到片上的数据通信调度以及 AIE 阵列计算。所有片上时间循环（即清单1中的第 6 行）的计算时间可由公式 7 定义，其中 MAC 表示单个 AIE 引擎在一个周期内可执行的理论乘加运算次数，Eff 表示计算内核实际达到的效率。这里我们同时考虑单个 AIE 和 AIE 阵列的流水线效率（PL↔AIE），并将整体效率设为 80%。对于片外到片上的调度，如清单1所述，最外层循环的循环顺序为 TY→TZ→TX，因此，左侧和右侧的内存访问总共会发生 TX×TX×TZ 次。总执行时间可通过公式8计算。这是一个用于说明的公式，我们省略了存储输出所花费时间以及流水线中的序言和尾声时间的具体计算细节。

$$Time_comp = (X \cdot Y \cdot Z \cdot TI \cdot TK \cdot TJ/MAC)/Eff \tag{7}$$

$$\begin{aligned} TIME = max([Time_L, Time_R, Time_comp]) \\ \cdot (TX \cdot TY \cdot TZ) \end{aligned} \tag{8}$$

对于任何特定的形状，所有可能的可配置参数都将以穷举方式进行评估。在完成 CDSE 后，将报告排名靠前的优化设计点。

5.4 CHARMDiverse Acc.Composer (CDAC)

CDAC 中的两步搜索算法。为了在多个加速器上映射不同规模的矩阵乘法内核时实现整体吞吐量优化，我们在 CDAC 中提出了一种基于排序的两步算法。在 CDAC 的第一步中，我们将输入模型中不同工作负载的矩阵乘法内核划分为多个组。组的数量等于异构加速器的数量，这是 CDAC 中的一个超参数。完成工作负载划分后，在第二步中，我们生成一个资源划分候选方案，规定为每个加速器分配的资源预算与其所分配的矩阵乘法内核中的操作总数成比例。在给定所分配的工作负载和资源的条件下，我们为每个加速器搜索可配置参数 (A,B,C,X,Y,Z) 的所有有效候选值。随后，我们对存储资源划分进行微调，以生成更多资源划分候选方案。完成内存微调后，我们生成新的工作负载划分，并重新执行资源划分和可配置参数搜索，从而进一步优化所有加速器的系统吞吐量。下面将详细讨论每个步骤。

第一步：工作负载分配。为了提高多样化加速器架构的整体吞吐量，我们需要将矩阵乘法内核合理地分配给各个加速器，并使它们并发工作且执行时间相近。然而，将包含 **n** 个内核的应用程序映射到 **num** 个加速器时，会面临指数时间复杂度，因为总映射搜索空间的规模为 $O(num^n)$。为了更好地扩展到包含更多内核（即 n 更大）的模型，我们提出一种基于排序的算法，以 $O(\binom{n-1}{num-1})$ 的时间复杂度 $= C_{num-1}^{n-1}$对工作负载进行分区。如算法1所示，CDAC 首先根据矩阵乘法内核的操作数对不同形状进行排序（第 4 行），以便合理划分规模较大的矩阵乘法和规模较小的矩阵乘法。随后，我们将排好序的矩阵乘法内核划分为 n 组（第 5–6 行）。例如，若有八种不同形状的内核需要映射到 num=2 个加速器，则对内核排序后，我们在任意两个内核之间放置一个分隔符，将所有内核分成两组。总共可得到$C\binom{8-1}{2-1} = 7$ 种分组设计方案。

第二步：硬件资源划分。对于每种工作负载分配，我们都会进行设计空间探索，在划分后的硬件资源约束条件下，找到优化后的加速器可配置参数，

算法1多样化加速器组合算法

输入：layer[n], bw，hw_sr，num，ubound

▷ **层[n]** 表示应用程序中有 n 个层。**带宽** 表示带宽，**硬件_资源** 包括 AIE、PLIO、随机存取存储器资源，**num** 表示加速器的数量，**ubound** 是内存调优的超参数

输出：工作负载[编号], 最终_加速器[编号]

▷ **工作负载** 和**最终_加速器**包含工作负载分配和硬件配置分别适用于每个加速器

1: 带宽 ←带宽/编号 $_{acc}$
2: 硬件 .随机存取存储器 [:] ← 硬件_sr.随机存取存储器/编号_加速器
3: 最终_周期 ←无穷
4: 层 _排序 [:] ← 排序(层)
5: **遍历**调度方案 in range($C_{n-1\ num-1}$) **执行**
6: 分区 [:] ← 分区(层_排序 [:], num, 调度方案) ▷ 1步
7: op_部分 [:] ←计数(分区 [:]) ▷ 2nd 步
8: 更新(硬件 .AIE [:], 硬件 .$PLIO$ [:], 操作_部分 [:])
9: 加速器 [:], 周期 [:] ← 加速器_搜索(硬件、带宽、分区 [:])
10: ▷ 顺序启动 CDSE
11: **当**调优计数 ≠ 上界 **时执行**_ ▷ 内存调优
12: 索引 ←最大值(周期 [:])
13: 更新(硬件。随机存取存储器 [:], 索引) ▷增加最慢加速器的存储器
14: 加速器 [:], 周期 [:] ← 加速器_搜索(硬件、带宽、分区 [:])
15: **如果**最大值(周期 [:]) < 最终_周期 **则** ▷ 更新最优点
16: 最终_周期 ←最大值(周期 [:])
17: 最终_加速器 [:] ←加速器 [:])
18: 工作负载 [:] ← 分区 [:]
19: 调优计数++_
20: 定义加速器_搜索(硬件，带宽，分区 [:]) :
21: **对于**范围(num) 中的加速器**执行**
22: CDSE(分区 [加速器], 硬件 [加速器],带宽 [加速器])
23: 返回 加速器 [:],周期 [;]

包括 AIE、PLIO、片上 RAM 和片外带宽的数量。为最大限度地缩短所有加速器中的最长执行时间，CDAC 会根据分配给加速器的操作总数，按比例分配 AIE 数量并设置 PLIO 约束（第 7–8 行）。对于片上 RAM 的数量，我们首先进行均匀分配（第 2 行）。依次启动 CDSE，为每个加速器寻找一次配置后（第 9–10 行），我们会进行内存微调，以优化内存分配。该步骤会找出耗时最长的加速器的索引（第 12 行），然后尝试通过增加该加速器的内存分配、同时减少其他加速器的内存分配，来探索更优的配置（第 13–14 行）。如果找到更优结果，我们就会更新全局最优执行周期以及对应的加速器配置设置（第 15–18 行）。请注意，在当前模型中，我们假设每个加速器均匀占用片外带宽（第 1 行），并将片外带宽划分问题留待未来研究。

5.5 CHARM 自动代码生成(CACG)

在从 CDAC 找到优化设计的硬件设计参数后，我们实现 CACG（包括 AIEGen、PLGen 和 HostGen），以生成对应的 AIE、可编程逻辑和主机 CPU 源代码。AIEGen 将单个 AIE 的分块因子 (TI,TK,TJ) 和 AIE 阵列 (A,B,C) 作为输入，并实例化相应数量的 AIE 核心。它利用基于 C++ 的自适应数据流 (ADF) Graph API [44]，通过 AXI 网络构建 AIE 核心之间的连接，并通过 PLIO 构建 AIE 阵列与可编程逻辑之间的连接。PLGen 根据可编程逻辑层级的 (X,Y,Z) 设计参数生成 HLS C/C++ 代码，在可编程逻辑侧分配片上缓冲区，并实现用于向 AIE 阵列发送数据及从中接收数据的数据传输模块。HostGen 则生成基于 Xilinx 运行时库 (XRT) API 的主机代码。

代码生成后，CHARM 会启动厂商工具，包括 AIE 编译器和 V++ 编译器，以生成输出目标文件 libadf.a 和 kernel.xo，并将其链接为一个 xclbin，即该设计的硬件位流。GCC 编译器会编译基于 XRT API 的主机代码，生成运行在 ARM CPU 上的主机程序，用于内核调度和系统控制。

5.6 CHARM 运行时调度器 (CRTS)

算法2 运行时调度算法

输入：图、数量、任务_池[任务][层]
输出： 每个加速器的运行时调度

1: **当(1) 时** ▷ 将就绪任务分配给相应的 Acc
2: **对** 加速器 属于范围(数量)**执行**
3: **如果** $\neg Acc[acc]$.空闲() **则**
4: **继续**
5: **对于** l 在 range(任务) **执行**
6: **对于** l 在 range(层) **执行**
7: **如果** 任务_池 $[t][l] \neq \emptyset \wedge$ 任务_池 $[t][l]$.有效() **则**
8: Acc[加速器].分配(任务_池 $[t][l]$)
9: **继续**第 2 行
10: **当** (1)**时** ▷ 根据依赖图更新任务_池
11: **遍历** 加速器 ， 范围为 range(num) **执行**
12: **如果**加速器[加速器].完成() **则**
13: $task_pool$.更新(图)
14: 加速器[加速器].更新(空闲)

为在每个任务内部的依赖约束条件下实现高吞吐量，同时保持相对较低的延迟，我们提出了 CRTS，它在运行时运行于 ARM CPU 上。算法2列出了调度算法。它以依赖图、加速器数量以及由 CDAC 生成的层分配配置文件作为输入。CRTS 中有两个并行进程。

第一个进程持续检查是否有可分配任务的空闲加速器（第 2–3 行）。CRTS 按照先进先出原则遍历分配给该加速器的各层（第 5–6 行）。如果某层仍在任务池中，说明该层尚未发出。假设当前层的所有前置层均已执行完毕，即依赖关系已解除。在这种情况下，CRTS 会将该有效层分配给相应的加速器（第 7–8 行），并继续检查其他加速器（第 9 行）。第二个进程持续跟踪每个加速器的状态，以检查其是否已完成工作负载（第 12–13 行），并根据依赖图更新任务池，同时将加速器的状态更改为空闲（第 14 行）。

6 实验结果

本节首先在第 6.1 节和第 6.2 节分别展示单个 AIE 的效率和单个矩阵乘法加速器的吞吐量。在第 6.3 节中，我们针对四种应用程序——BERT、ViT、NCF 和多层感知机（MLP）——实现了不同的 CHARM 设计，包括一个单体矩阵乘法加速器、一个专用矩阵乘法加速器、两个多样化矩阵乘法加速器以及八个复制型矩阵乘法加速器。所有实验均在 VCK190 上进行，可编程逻辑（PL）运行频率为 230 MHz，AIE 运行频率为 1 GHz。我们使用 AMD/Xilinx Vitis 2021.1 版作为编译后端工具。测量功耗时，我们让每个应用程序运行超过 60 秒，并使用板卡评估与管理工具 AMD/Xilinx BEAM [45] 报告平均值。

6.1 单个 AIE 内核效率比较

本节展示了单个 AIE 在不同单精度浮点矩阵尺寸下的矩阵乘法计算效率。我们利用 AIE内建函数 [46] 对单个内核进行编程，并通过仿真获得

表 2：fp32 数据类型下单个 AIE 矩阵乘法的比较。

尺寸：M x K x N	H-GCN[48] 每周期乘加次数	H-GCN[48] Eff	CHARM（本研究） 每周期乘加次数	CHARM（本研究） Eff	效率提升
16 × 16 × 16	2.34	29.30%	6.18	77.22%	***2.64x***
32 × 32 × 32	3.64	45.50%	7.57	94.70%	***2.08x***
64 × 64 × 8	3.64	45.50%	7.54	94.29%	***2.07x***

表 3：不同矩阵尺寸下，板上测量与 CDSE 分析建模估算结果的性能对比（单位：每秒十亿次浮点运算）。误差率百分比表明，CDSE 具有较高的预测准确度。

方形矩阵乘法尺寸	板载估算		误差功耗（瓦）	
64	0.41	0.40	-2%	32.58
128	3.36	3.22	-4%	32.86
256	25.58	25.79	1%	34.66
512	176.24	178.42	1%	37.95
1024	1103.46	1123.81	2%	41.78
1536	1633.13	1649.01	1%	46.02
2048	1672.76	1688.17	1%	47.87
3072	2850.13	2895.90	2%	50.65
4096	2718.42	2773.26	2%	51.97
6144	3277.99	3363.89	3%	53.57

单个 AIE 设计的执行周期数。仿真采用 Versal ACAP AI 引擎 System C 仿真器 [47]，这是一款周期精确的架构仿真器。如表 2 所示，当矩阵乘法规模等于 32×32×32 时，单个 AIE 的性能最高可达每周期乘加运算次数 7.57，达到峰值性能的 94.70%。与 H-GCN [48]，报告的 AIE 稠密矩阵乘法内核效率相比，我们的单个内核平均效率提升了 2.26×。对于整个系统设计，我们选择 32×32×32作为单个内核，因为它计算效率高，且左侧、右侧和输出矩阵的总尺寸不超过 16 KB，因此可以放入 AIE 本地存储器并采用双缓冲。

6.2 一个单体加速器上的方阵矩阵乘法性能

我们评估了一个单体加速器设计的吞吐量，并比较了 CDSE 建模估算与板上测量的性能。我们使用 384 个 AIE，并将片上 RAM 利用率提高到 83% 以上，构建了该单体设计；AIE 的运行频率为 1 GHz，可编程逻辑侧的运行频率为 230 MHz。如表 3 所示，随着方形矩阵乘法规模增大，单加速器单体设计的吞吐量也随之提升。规模为 6144 时，其吞吐量达到 3.27 太浮点运算每秒；而规模为 64 时，吞吐量仅为 0.41 每秒十亿次浮点运算。CHARM CDSE 能够准确估算板上执行时间，平均估算误差率仅为 2.9%。

我们比较了采用最先进的脉动阵列框架 AutoSA [22] 、针对 fp32 数据类型实现的相同矩阵乘法应用程序在 VCK190 可编程逻辑部分中的吞吐量。VCK190 的可编程逻辑侧配备了 1968 个 DSP58 IP 核。上一代开发板使用五个 DSP48 执行浮点乘法，而该板只需一个 DSP58。

如表 4 所示，CHARM 单矩阵乘法加速器的吞吐量达到 3.27 TFLOPs；与 VCK190 上仅使用可编程逻辑的设计相比，其吞吐量提升了 5.54× 倍，能效提升了 1.93× 倍。

表 4：CHARM 中 VCK190 上仅使用可编程逻辑的设计与采用可编程逻辑存储器 + AIE 的设计对比。

数据类型	可编程逻辑 [22] Float32	CHARM Float32
频率	可编程逻辑：200MHz	可编程逻辑：230MHz
URAM	0	384
BRAM	923	764
	DSP58:1536	AIE:384
TFLOPs	0.59 (1x)	3.27 (5.54x)
功率（瓦）	18.60	53.40
能效	1x	1.93x

表 5：BERT、ViT、NCF、多层感知机中的矩阵乘法尺寸。

模型	层数	M	K	N	批量点大小
BERT	4	3072	1024	1024	N/A
	1	3072	4096	1024	N/A
	1	3072	1024	4096	N/A
	1	512	64	512	96
	1	512	512	64	96
ViT	1	3072	3024	1024	N/A
	1	3072	1024	1024	N/A
	1	3072	1024	4096	N/A
	1	3072	4096	1024	N/A
	1	3072	1024	3048	N/A
	2	64	64	64	768
NCF	1	3072	4096	2048	N/A
	1	3072	2048	1024	N/A
	1	3072	1024	512	N/A
	1	3072	512	256	N/A
	1	3072	256	128	N/A
	1	3072	128	64	N/A
	1	3072	64	32	N/A
	1	3072	32	16	N/A
	1	3072	32	1	N/A
MLP	1	3072	2048	4096	N/A
	2	3072	4096	4096	N/A
	1	3072	4096	1024	N/A

表 6：在 BERT、ViT、NCF 和多层感知机的端到端解决方案中，实现最高吞吐量的不同类型内核的耗时细分。

内核	BERT	ViT	NCF	MLP
MM	57.2毫秒	57.7毫秒	40.4毫秒	11.9毫秒
层归一化	4.5毫秒	4.5毫秒	0	0
Softmax	18.7毫秒	2.3毫秒	0	0
转置	5.2毫秒	5.2 毫秒	0	0

6.3 端到端性能

我们将 CHARM 框架应用于四种应用程序：BERT、ViT、NCF 和多层感知机。这些模型中所有矩阵乘法内核的形状均列于表 5。我们考察了加速器数量从 1 到 8 的情况，并针对每种应用程序展示了具有代表性的 CHARM 设计，包括一个单体式矩阵乘法加速器、一个专用矩阵乘法加速器、两个多样化矩阵乘法加速器，以及八个复制型矩阵乘法加速器。单体式矩阵乘法设计见第 6.2 节，对四种应用程序均保持不变，并作为比较的基准设计。其余矩阵乘法加速器设计均针对各应用程序定制，并使用 CHARM 框架进行设计和实现。所有

表7：BERT、ViT、NCF 和多层感知机在不同矩阵乘法加速器配置下的板载吞吐量和功耗比较。

App	CHARM配置	LUT	BRAM	URAM	DSP	AIE	每秒十亿次浮点运算 功率（瓦）	每瓦每秒十亿次浮点运算（比率）	
BERT	一个单声道_	103959(11.55%)	764 (79.01%)	384 (82.94%)	165 (8.38%)	384 (96%)	276.8	37.0	7.48 (1x)
	One_spe	90351(10.04%)	515 (53.26%)	64 (13.82%)	117 (5.95%)	256 (64%)	515.4	32.4	15.91 (2.13 倍)
	两个多样化的_	***343774 (38.20%)***	***534 (55.22%)***	***272 (58.75%)***	***442 (22.46%)***	***288 (72%)***	***1464.2***	***40.7***	***35.98 (4.81x)***
	8_重复	222956(24.78%)	664 (68.67%)	384 (82.94%)	488（24.80%）	256（64%）	534.2	34.2	15.62（2.09 倍）
ViT	一个单声道_	103959（11.55%）	764（79.01%）	384 (82.94%)	165 (8.38%)	384 (96%)	49.5	32.4	1.53 (1x)
	One_spe	76661(8.52%)	275（28.44%）	64（13.82%）	187（9.50%）	256（66%）	217.1	28.0	7.75（5.08 倍）
	两个多样化的_	***240563(26.73%)***	***590 (61.01%)***	***320 (69.11%)***	***299 (15.19%)***	***264 (72%)***	***1609.0***	***39.6***	***40.63 (26.60x)***
	8_重复	222956(24.78%)	664 (68.67%)	384 (82.94%)	488 (24.80%)	256 (64%)	382.2	32.8	11.65 (7.63x)
NCF	一个单声道_	***103959(11.55%)***	***764 (79.01%)***	***384 (82.94%)***	***165 (8.38%)***	***384 (96%)***	***1736.0***	***45.2***	***38.41 (1x)***
	One_spe	***103959(11.55%)***	***764 (79.01%)***	***384 (82.94%)***	***165（8.38%）***	***384（96%）***	***1736.0***	***45.2***	***38.41（1.00x）***
	两个多样化的_	161597（17.96%）	790（81.70%）	352（76.03%）	326（16.57%）	384（96%）	1730.9	45.1	38.38（0.99 倍）
	8_重复	222956（24.78%）	664（68.67%）	384（82.94%）	488（24.80%）	256（64%）	671.0	35.0	19.17（0.50x）
MLP	一个单声道_	***103959(11.55%)***	***764 (79.01%)***	***384 (82.94%)***	***165 (8.38%)***	***384 (96%)***	***2936.7***	***51.4***	***57.13 (1x)***
	一个单声道_	***103959(11.55%)***	***764 (79.01%)***	***384 (82.94%)***	***165 (8.38%)***	***384 (96%)***	***2936.7***	***51.4***	***57.13 (1.00x)***
	两个多样化的_	148158(16.46%)	919 (95.04%)	448 (96.76%)	344 (17.48%)	384（96%）	2386.1	48.8	48.90（0.86x）
	8_重复	222956（24.78%）	664（68.67%）	384（82.94%）	488 (24.80%)	256 (64%)	696.0	35.2	19.77 (0.35x)

表8：采用两个多媒体多样化加速器和四个非多媒体加速器的 BERT 设计中各加速器的资源利用率。

Type	REG	查找表逻辑	查找表存储器	BRAM	URAM	DSP	AIE
矩阵乘法0+直接存储器访问0+缓冲区	96790 (5.55%)	91034 (10.41%)	835 (0.19%)	515 (53.26%)	256(55.29%)	246(12.50%)	256(64%)
矩阵乘法1+直接存储器访问1+缓冲区	62415 (3.58%)	94739（10.83%）	37668（8.48%）	19（1.96%）	16（3.46%）	196(9.96%)	32 (8%)
层归一化	45101 (2.58%)	33939 (3.88%)	4234 (0.95%)	15 (1.55%)	90 (19.44%)	129(6.55%)	0 (0%)
Softmax	34270 (1.96%)	33623 (3.84%)	2854 (0.64%)	243 (25.13%)	0 (0%)	151(7.67%)	0 (0%)
转置0	14217 (0.81%)	6926 (0.79%)	1097 (0.25%)	15 (1.55%)	0 (0%)	94 (4.78%)	0 (0%)
转置1	33967 (1.95%)	58510 (6.69%)	32512 (7.32%)	15 (1.55%)	0 (0%)	19 (0.97%)	0 (0%)

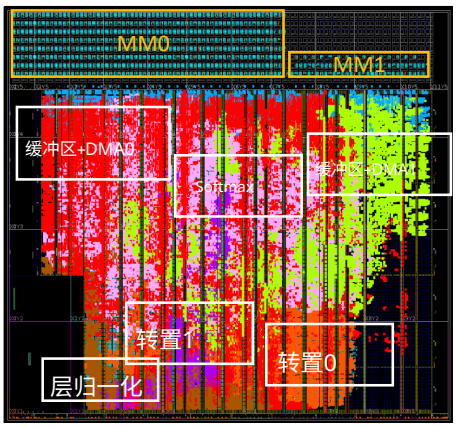

图7：BERT 的两个多样化矩阵乘法加速器和四个非矩阵乘法加速器的系统实现布局。

同一应用程序的设计均使用相同的非矩阵乘法内核。表7报告了四种应用程序在不同加速器配置下的板载吞吐量和功耗。

CHARM 为 BERT、ViT、NCF 和 MLP 中的矩阵乘法分别实现了 1.46 TFLOPs、1.61 TFLOPs、1.74 TFLOPs 和 2.94 TFLOPs 的最大吞吐量。表 6 展示了每个端到端应用中矩阵乘法、层归一化、Softmax 和转置的时间分解。我们在表7中标出了每个应用的最佳设计。在 BERT 和 ViT 中，采用两种多样化矩阵乘法加速器的设计表现最佳；而对于 NCF 和 MLP，采用单个加速器的设计表现最佳。这是因为 BERT 和 ViT 同时包含大型和小型矩阵乘法，而 MLP 仅包含大型矩阵乘法。NCF 同样同时包含大型和小型矩阵乘法。然而，小型

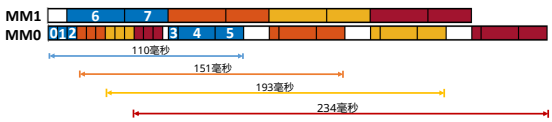

图8：BERT 的 4 个任务在 2 个加速器上的调度时间线。

矩阵乘法消耗的总计算量不到 0.8%，因此偏重大型矩阵乘法的设计表现最佳。对于所有应用，八重复设计由于每个加速器的数据复用不足而表现较差。

对于 BERT 和 ViT，与单体设计相比，CHARM 针对特定应用提供的专用加速器设计分别使能效（每瓦每秒十亿次浮点运算）提升了 2.13× 和 5.08×。通过使用 CHARM 框架提供的多个异构且形状各异的加速器，探索额外的设计空间，使 BERT 和 ViT 的能效分别进一步提升了 2.25× 和 5.24×。**这些提升展现了 CHARM 创新的设计方法，即组合异构加速器。**

图7展示了两种异构矩阵乘法加速器设计的实现布局，也就是 BERT 的最佳设计。这也是图5所示的布局，其中包含两个矩阵乘法加速器和四个非矩阵乘法、受通信限制的加速器。表8报告了每个加速器的硬件资源利用率。矩阵乘法加速器0利用256个 AIE、53.26% 的块随机存取存储器和55.29%的超大容量存储器，在计算大型矩阵乘法时实现了较高的数据复用率和计算效率。矩阵乘法加速器1利用 32个 AIE、1.96%的块随机存取存储器和3.46%的超大容量存储器，能够满足 BERT 中小型矩阵乘法所需的计算和通信需求，同时避免资源过度配置。

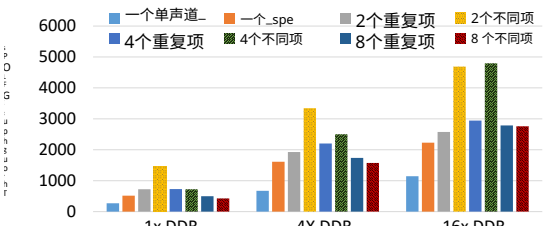

图 9：CHARM 在不同片外带宽配置下的 BERT 吞吐量对比。

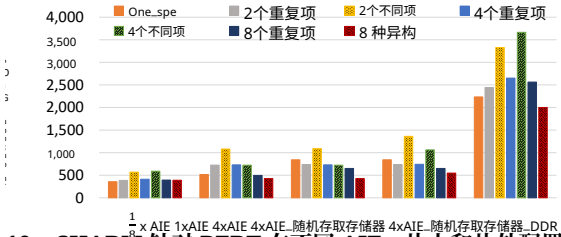

图 10：CHARM 针对 BERT 在不同 AIE、片上和片外配置下的吞吐量比较。

探索 CHARM 中的延迟-吞吐量权衡。如图8所示，我们在包含两种异构加速器的 BERT 设计上映射了四个并发任务。每个任务有八个矩阵乘法内核，并且存在依赖边，包括 0→6, 1→6, 6→7, 2→7, 7→3→4→5，其中 x→y 表示 y 依赖于 x。为使图示更简洁，图中未显示其他受通信限制的非矩阵乘法内核。完成第1个任务需要110 ms，完成第4个任务需要234ms。对于单加速器专用设计，每个任务的延迟为162.6ms。因此，我们面临设计权衡：单个专用加速器设计可以处理细粒度任务，而两种异构加速器设计则需要粗粒度任务来填满两个加速器的流水线。与单个专用加速器设计相比，在不同任务的延迟为0.67×，
0 92× 1 18× 1 43× 2 8× 。 , . , . 的情况下，我们换取了 . 的总体吞吐量提升。这表明 CHARM 框架支持探索延迟-吞吐量权衡，用户可以指定目标，让 CHARM 生成能够在满足延迟要求的同时优化吞吐量，或反之的设计。

CHARMDSE 效率。我们在 CDAC 中使用 CHARM 执行基于排序的两步搜索算法。对于 BERT，与穷举搜索相比，CDAC 可在 170 秒内找到最优解，而穷举搜索需要 33 分钟（搜索迭代次数：200 万次 vs. 5800 万次）；测试使用 MATLAB R2021b，在 Intel Core i9-10900X CPU 上进行。

7 架构洞见与映射洞见讨论

借助 CHARM 框架提供的强大建模能力，我们探索了不同硬件架构变更（AIE 数量、片上存储容量、片外带宽）下的性能，以开展流片前架构探索，并提供有助于未来一代设备的架构设计洞见。在此，我们利用 CHARM 建模结果报告系统中不同加速器配置（包括 1、2、4 和 8 个加速器）的吞吐量。对于每种设计（单加速器设计除外），我们提供两种变体：同构加速器或异构加速器。这些探索有助于我们理解以下研究问题：

问题 1：我们能否从更高的片外带宽中获益？回答 1：能。Versal 需要更高的片外带宽。
我们首先探究性能，假设平台拥有更多
片外带宽。我们将 DDR 带宽提高 4×，以模拟多个 DDR 存储体，并提高 16×，以模拟使用高带宽存储器（HBM）的情况。如图 9 所示，在不同带宽配置下，BERT 的最佳设计的吞吐量从 1.48 TFLOPs 提升至带宽为 4× 时的 3.34 TFLOPs，以及带宽为 16× 时的 4.80 TFLOPs。从 1× 到 4× DDR 的性能提升符合预期，这意味着

BERT 的设计受片外带宽限制。 16× 的最大吞吐量受系统计算吞吐量限制，为 4.8 TFLOPs；该吞吐量受单内核计算效率（95%）和可编程逻辑↔AIE 效率（85%）的制约。从图 9 还可以观察到，多个加速器的吞吐量提升幅度大于单个加速器，因为加速器数量增加时，每个加速器的数据复用会减少，因此更容易受到片外带宽的限制。

Q2：我们能否在未来的架构中利用 CHARM？A2：可以。图 10 中的最后一组结果表明，随着未来计算和通信并行性进一步提高，我们需要更多异构加速器架构，而 CHARM 可作为最有前景的解决方案之一。

我们通过改变 AIE 数量、片上随机存取存储器容量和片外带宽来研究性能。我们将 AIE 数量减少到当前 AIE 阵列规模的 1/8，以模拟上一代 FPGA 的计算能力；上一代 FPGA 仅在可编程逻辑中配备数字信号处理器，其单精度浮点理论峰值性能约为 Versal ACAP 的 1/8。如图 10 中的第一组所示，不同加速器配置下的最低性能与最高性能相差不到 40%。随着计算并行性降至 1/8，大规模并行性与较小矩阵乘法规模不匹配所造成的资源浪费得以缓解。另一方面，如图 10 中的最后一组所示，当我们将 AIE、片上存储和片外带宽都提升 4× 时，4 种异构加速器表现最佳。单纯增加 AIE 数量并不能显著提升性能，而整体增加所有资源则可以。

8 结论与致谢

本文提出了 CHARM 架构和 CHARM 框架，为在应用程序中组合用于不同矩阵乘法的异构加速器并生成端到端应用解决方案，提供了一种新颖的系统级设计方法。未来，我们将探索并扩展 CHARM，使其适用于更多应用和更多数据类型。

我们感谢匹兹堡大学新教师启动基金、美国国家科学基金会项目 #2213701 、 #2217003，以及 CRISP 的支持。CRISP 是 JUMP 的六个中心之一。我们感谢所有审稿人提供的宝贵反馈，并感谢 Marci Baun 协助编辑论文。我们感谢 AMD/Xilinx 捐赠 FPGA 和软件，并感谢伊利诺伊大学厄巴纳-香槟分校的 AMD/Xilinx 卓越中心、加利福尼亚大学洛杉矶分校的 AMD/Xilinx 异构加速计算集群，以及匹兹堡大学研究计算中心（CRC）提供的支持。

参考文献

[1] Ashish Vaswani、Noam Shazeer、Niki Parmar、Jakob Uszkoreit、Llion Jones、Aidan N Gomez、Łukasz Kaiser 和 Illia Polosukhin。Attention Is All You Need。神经信息处理系统进展，30，2017年。[2] Xiangnan He、Lizi Liao、Hanwang Zhang、Liqiang Nie、Xia Hu 和 Tat-Seng Chua。神经协同过滤。载于第 26 届国际万维网大会论文集，第 173–182 页，2017年。[3] Alexey Dosovitskiy、Lucas Beyer、Alexander Kolesnikov、Dirk Weissenborn、Xi- aohua Zhai、Thomas Unterthiner、Mostafa Dehghani、Matthias Minderer、Georg Heigold、Sylvain Gelly 等。图像胜过千言万语：面向大规模图像识别的 Transformer。arXiv 预印本 arXiv:2010.11929，2020年。[4] Yu Emma Wang、Gu-Yeon Wei 和 David Brooks。深度学习平台张量处理器、图形处理器和中央处理器的基准测试。arXiv 预印本 arXiv:1907.10701，2019年。[5] Norman P Jouppi、Cliff Young、Nishant Patil、David Patterson、Gaurav Agrawal、Raminder Bajwa、Sarah Bates、Suresh Bhatia、Nan Boden、Al Borchers 等。张量处理器在数据中心内的性能分析。载于第 44 届年度国际计算机体系结构研讨会论文集，第 1–12 页，2017年。[6] Onur Mutlu、Saugata Ghose、Juan Gómez-Luna 和 Rachata Ausavarungnirun。存内处理现代入门。载于《新兴计算：从器件到系统》，第 171–243 页。施普林格，2023年。[7] Geraldo F Oliveira、Juan Gómez-Luna、Lois Orosa、Saugata Ghose、Nandita Vi-jaykumar、Ivan Fernandez、Mohammad Sadrosadati 和 Onur Mutlu。DAMOV：用于评估数据移动瓶颈的新方法与基准测试套件。IEEE Access，9：134457–134502，2021年。[8] Hasan Hassan、Minesh Patel、Jeremie S Kim、A Giray Yaglikci、Nandita Vijayku-mar、Nika Mansouri Ghiasi、Saugata Ghose 和 Onur Mutlu。Crow：一种低成本基底，可提升动态随机存取存储器的性能、能效和可靠性。载于第 46 届国际计算机体系结构研讨会论文集，第 129–142 页，2019年。[9] Jim Demmel。通信规避算法。载于《2012 SC Companion：高性能计算、网络、存储与分析》，第 1942–2000 页。电气电子工程师学会，2012年。[10] AMD/Xilinx。Versal 自适应计算加速平台。[11] AMD。深度学习处理单元 IP 覆盖层，2022年。[12] Yu-Hsin Chen 等。Eyeriss：面向卷积神经网络的能效数据流空间架构。ACM SIGARCH 计算机体系结构新闻，2016年。[13] Yu-Hsin Chen、Tien-Ju Yang、Joel Emer 和 Vivienne Sze。Eyeriss v2：面向移动设备上新兴深度神经网络的灵活加速器。IEEE 电路与系统新兴及精选主题期刊，9(2)：292–308，2019年。[14] Zidong Du、Robert Fasthuber、Tianshi Chen、Paolo Ienne、Ling Li、Tao Luo、Xiaob-ing Feng、Yunji Chen 和 Olivier Temam。ShiDiannao：将视觉处理移至更靠近传感器的位置。载于第 42 届年度国际计算机体系结构研讨会论文集，第 92–104 页，2015年。[15] Eriko Nurvitadhi、Dongup Kwon、Ali Jafari、Andrew Boutros、Jaewoong Sim、Phillip Tomson、Huseyin Sumbul、Gregory Chen、Phil Knag、Raghavan Kumar 等。既然可以协同工作，何必竞争：用于持久化循环神经网络的 FPGA-ASIC 集成。载于 2019 年电气电子工程师学会第 27 届年度国际可定制计算机研讨会（FCCM），第 199–207 页。电气电子工程师学会，2019年。[16] Andrew Boutros、Eriko Nurvitadhi、Rui Ma、Sergey Gribok、Zhipeng Zhao、James C Hoe、Vaughn Betz 和 Martin Langhammer。超越峰值性能：比较 AI 优化 FPGA 与图形处理器的实际性能。载于 2020 年国际可编程技术会议（ICFPT），第 10–19 页。电气电子工程师学会，2020年。[17] Jeremy Fowers、Kalin Ovtcharov、Michael Papamichael、Todd Massengill、Ming Liu、Daniel Lo、Shlomi Alkalay、Michael Haselman、Logan Adams、Mahdi Ghandi 等。面向实时人工智能的可配置云规模深度神经网络处理器。载于 2018 年 ACM/电气电子工程师学会第 45 届年度国际计算机体系结构研讨会（ISCA），第 1–14 页。电气电子工程师学会，2018年。[18] Tiziano De Matteis、Johannes de Fine Licht 和 Torsten Hoefler。FBLAS：在 FPGA 上进行流式线性代数运算。载于 SC20：高性能计算、网络、存储与分析国际会议，第 1–13 页。电气电子工程师学会，2020年。[19] Johannes de Fine Licht、Grzegorz Kwasniewski 和 Torsten Hoefler。采用高级综合技术在 FPGA 上实现灵活的通信规避矩阵乘法。载于 2020 年 ACM/SIGDA 可编程门阵列国际研讨会论文集，第 244–254 页，2020年。[20] Chen Zhang 等。优化基于 FPGA 的加速器设计，以用于深度卷积神经网络。载于 FPGA 会议论文集，第 161–170 页。ACM，2015年。[21] Duncan J. M. Moss、Srivatsan Krishnan、Eriko Nurvitadhi、Piotr Ratuszniak、Chris Johnson、Jaewoong Sim、Asit Mishra、Debbie Marr、Suchit Subhaschandra 和 Philip H. W. Leong。面向 Intel Harp v2 Xeon+FPGA 平台的可定制矩阵乘法框架：深度学习案例研究。载于 2018 年 ACM/SIGDA 可编程门阵列国际研讨会论文集，FPGA '18，第 107–116 页。美国计算机协会，2018年 2 月。[22] Jie Wang、Licheng Guo 和 Jason Cong。AutoSA：一种面向 FPGA 高性能脉动阵列的多面体编译器。发表于 2021 年 ACM/SIGDA 可编程门阵列国际研讨会，FPGA '21，第 93–104 页。美国计算机协会，2021 年 2 月。[23] Linghao Song、Yuze Chi、Atefeh Sohrabizadeh、Young-kyu Choi、Jason Lau 和 Jason Cong。Sextans：一种用于通用稀疏矩阵-稠密矩阵乘法的流式加速器。发表于 2022 年 ACM/SIGDA 可编程门阵列国际研讨会论文集，FPGA '22，第 65–77 页，美国纽约州纽约市，2022 年。美国计算机协会。[24] Linghao Song、Yuze Chi、Licheng Guo 和 Jason Cong。Serpens：一种基于高带宽存储器的通用稀疏矩阵-向量乘法加速器。发表于第 59 届 ACM/IEEE 设计自动化会议论文集，第 211–216 页，2022 年。[25] Jason Cong、Peng Wei、Cody Hao Yu 和 Peipei Zhou。Latte：面向高级综合的局部性感知变换。发表于 2018 年 IEEE 第 26 届年度国际现场可编程定制计算机研讨会（FCCM），第 125–128 页，2018 年。[26] Peipei Zhou、Hyunseok Park、Zhenman Fang、Jason Cong 和 André DeHon。全流水线的能效：矩阵乘法案例研究。发表于 2016 年 IEEE 第 24 届年度国际现场可编程定制计算机研讨会（FCCM），第 172–175 页，2016 年。[27] Peipei Zhou、Jiayi Sheng、Cody Hao Yu、Peng Wei、Jie Wang、Di Wu 和 Jason Cong。MOCHA：在异构云中对加速器进行多节点成本优化。发表于 2021 年 ACM/SIGDA 可编程门阵列国际研讨会，FPGA '21，第 273–279 页，美国纽约州纽约市，2021 年。美国计算机协会。[28] Xiaofan Zhang 等。DNNBuilder：一种用于构建高性能 FPGA 深度神经网络硬件加速器的自动化工具。发表于 Proc. I CCAD，第 56 页。ACM，2018 年。[29] Xiaofan Zhang、Hanchen Ye、Junsong Wang、Yonghua Lin、Jinjun Xiong、Wen-mei Hwu 和 Deming Chen。DNNExplorer：用于建模和探索一种新型 FPGA 深度神经网络加速器范式的框架。发表于第 39 届国际计算机辅助设计会议论文集，第 1–9 页，2020 年。[30] Mingyu Gao、Jing Pu、Xuan Yang、Mark Horowitz 和 Christos Kozyrakis。Tetris：利用 3D 存储器实现可扩展、高效的神经网络加速。发表于第二十二届国际编程语言与操作系统体系结构支持会议论文集，第 751–764 页，2017 年。[31] Mingyu Gao、Xuan Yang、Jing Pu、Mark Horowitz 和 Christos Kozyrakis。Tangram：面向可扩展神经网络加速器的优化粗粒度数据流。发表于第二十四届国际编程语言与操作系统体系结构支持会议论文集，第 807–820 页，2019 年。[32] Hyoukjun Kwon、Liangzhen Lai、Michael Pellauer、Tushar Krishna、Yu-Hsin Chen 和 Vikas Chandra。面向多深度神经网络工作负载的异构数据流加速器。发表于 2021 年 IEEE 高性能计算机体系结构国际研讨会（HPCA），第 71–83 页。IEEE，2021 年。[33] Nvidia。网站。http://nvdla.org/。[34] Jason Cong、Hui Huang、Chiyuan Ma、Bingjun Xiao 和 Peipei Zhou。全流水线、动态可组合的 CGRA 架构。发表于 2014 年 IEEE 第 22 届年度国际现场可编程定制计算机研讨会，第 9–16 页，2014 年。[35] Jason Cong、Mohammad Ali Ghodrat、Michael Gill、Beayna Grigorian 和 Glenn Reinman。CHARM：一种可组合的异构加速器丰富型微处理器。发表于 2012 年 ACM/IEEE 低功耗电子与设计国际研讨会论文集，ISLPED '12，第 379–384 页，美国纽约州纽约市，2012 年。美国计算机协会。[36] AMD/Xilinx。Versal AI Core 系列 VCK190 评估套件，2022 年。[37] AMD/Xilinx。AI 引擎技术，2022 年。[38] Jason Cong、Bin Liu、Stephen Neuendorffer、Juanjo Noguera、Kees Vissers 和 Zhiru Zhang。FPGA 高级综合：从原型设计到部署。《IEEE 计算机辅助设计集成电路与系统汇刊》，30(4)：473–491，2011 年。[39] Jason Cong、Jason Lau、Gai Liu、Stephen Neuendorffer、Peichen Pan、Kees Vissers 和 Zhiru Zhang。今日 FPGA 高级综合：成功、挑战与机遇。《ACM 可重构技术与系统汇刊》（TRETS），15 42，2022 4 1( ): – 。[40] Alexandros Papakonstantinou、Karthik Gururaj、John A Stratton、Deming Chen、Jason Cong 和 Wen-Mei W Hwu。FCUDA：实现 CUDA 内核向 FPGA 的高效编译。发表于 2009 年 IEEE 第 7 届应用专用处理器研讨会，第 35–42 页。IEEE，2009 年。[41] Alexandros Papakonstantinou、Yun Liang、John A Stratton、Karthik Gururaj、Deming Chen、Wen-Mei W Hwu 和 Jason Cong。FPGA 上的多级粒度并行综合。发表于 2011 年 IEEE 第 19 届年度国际现场可编程定制计算机研讨会，第 178–185 页。IEEE，2011 年。[42] Yun Liang、Kyle Rupnow、Yinan Li、Dongbo Min、Minh N Do 和 Deming Chen。高级综合：生产力、性能与软件约束。《电气与计算机工程期刊》，2012 年，2012 年。[43] Yuze Chi、Licheng Guo、Jason Lau、Young-kyu Choi、Jie Wang 和 Jason Cong。扩展高级综合以支持任务并行程序。发表于 2021 年 IEEE 第 29 届年度

国际现场可编程定制计算机研讨会 *(FCCM)*，第 204–213 页，2021 年。
[44] AMD/Xilinx。自适应数据流 API。[45] AMD/Xilinx。板卡评估与管理工具。
[46] AMD/Xilinx。AI 引擎 API 与内置函数用户指南。
[47] AMD/Xilinx。VersalTM ACAP AI 引擎 System C 仿真器。[48] 张成明、耿通、郭安琪、田建南、Martin Herbordt、李昂和陶丁文。H-GCN：Versal ACAP 架构上的图卷积网络加速器。*arXiv* 预印本 *arXiv:2206.13734*，2022 年。