中国科学技术大学

本科毕业论文

动态可重构的 FPGA 矩阵乘法加速系统设计

作者姓名:	张学涵
学　　号:	PB21000079
专　　业:	计算机科学与技术
导　　师:	宫磊 教授
完成时间:	2025 年 5 月 12 日

摘　要

摘要是论文内容的总结概括，应简要说明论文的研究目的、基本研究内容、研究方法或过程、结果和结论，突出论文的创新之处。摘要应具有独立性和自明性，即不用阅读全文，就能获得论文必要的信息。摘要中不宜使用公式、图表，不引用文献。

摘要分中文和英文两种，中文在前，英文在后，内容及段落须相互呼应。博士论文中文摘要一般 800～1000 个汉字，硕士论文中文摘要一般 600 个汉字。英文摘要的篇幅参照中文摘要。

论文的关键词，是为了文献标引工作从论文中选取出来用以表示全文主题内容信息的单词或术语。建议关键词数量不超过 8 个，每个关键词之间用分号间隔。

英文摘要部分的标题为“ABSTRACT”。每个关键词第一个字母大写，关键词之间用半角逗号加空一格间隔，英文关键词与中文关键词须相互呼应。

关键词：学位论文；摘要；关键词

ABSTRACT

The length of the English abstract should refer to that of the Chinese abstract. The title of the English abstract is “ABSTRACT”. The first letter of each keyword should be capitalized, and keywords should be separated by a halfwidth comma and a space. The English keywords and Chinese keywords should correspond to each other.

Key Words: Dissertation; Abstract; Keywords

目　录

第一章　绪论

第一节　研究背景与意义

我们正处在一个信息爆炸的时代，以大数据、人工智能（AI）、物联网（IoT）等为代表的新兴技术蓬勃发展，驱动着社会各领域的深刻变革 [1, 2]。在这些技术的背后，海量数据的处理和分析是核心环节，而矩阵运算，特别是矩阵乘法，作为一种基础且计算密集型的操作，广泛应用于科学计算、图像处理、信号处理、机器学习（尤其是深度学习）、推荐系统、图计算等众多领域 [3, 4]。例如，在深度学习中，神经网络的训练和推理过程涉及大量的卷积和全连接层计算，这些本质上都可以归结为大规模的矩阵或张量运算 [5]。在图计算中，邻接矩阵的乘法可用于发现节点间的路径关系 [6]。随着模型复杂度和数据规模的持续增长，对计算能力的需求也呈现出指数级增长的态势，传统的计算模式面临着严峻的挑战。

中央处理器（CPU）作为通用的计算核心，虽然具有强大的逻辑控制能力和灵活性，但在处理高度并行化的数据密集型任务（如大规模矩阵乘法）时，其固有的串行执行特性和有限的并行处理单元（核心数）往往导致性能瓶颈 [7]。图形处理器（GPU）凭借其众多的计算核心（流处理器）和高内存带宽，在并行计算领域取得了巨大成功，特别是在稠密矩阵运算和深度学习领域展现出卓越的加速效果 [8, 9]。然而，GPU 的体系结构相对固定，对于某些特定类型的计算（例如，具有高度不规则访存模式的稀疏矩阵运算）或者需要细粒度流水线定制的任务，其效率可能并非最优 [10]。此外，GPU 通常功耗较高，且其编程模型（如 CUDA 或 OpenCL）虽然强大，但与底层硬件的映射关系不如 FPGA 直接。

现场可编程门阵列（FPGA）作为一种可编程硬件，提供了一种介于通用处理器和专用集成电路（ASIC）之间的解决方案 [11]。FPGA 内部包含大量的可配置逻辑块（CLB）、存储单元（BRAM）、DSP 单元等，用户可以通过硬件描述语言（HDL）或高层次综合（HLS）工具对其进行编程，实现高度定制化的硬件加速器 [12]。相比 CPU，FPGA 能够实现真正在硬件层面的并行和流水线处理，大幅提升计算密集型任务的性能功耗比。相比 GPU，FPGA 的架构灵活性允许针对特定算法进行深度优化，例如为稀疏矩阵的不同存储格式（如 CSR, CSC, COO 等）设计专门的处理单元和访存逻辑 [13]。相比 ASIC，FPGA 虽然在绝对性能和功耗上可能稍逊一筹，但其可重复编程的特性大大降低了设计成本和风险，缩

短了开发周期，尤其适合算法快速迭代或需要适应多种应用场景的领域。

然而，传统的基于 FPGA 的加速器设计通常是“静态”的，即一旦配置了 FPGA，其硬件逻辑就固定下来，直到下一次完全重新配置。这种模式对于功能固定的应用是有效的，但在许多现代应用场景中，计算任务的需求可能是动态变化的。例如，一个复杂的数据处理流程可能包含多个阶段，每个阶段需要不同的加速核心；或者，系统需要处理不同格式、不同稀疏度的稀疏矩阵，为每种情况设计一个最优的静态加速器并在需要时切换，会导致整个 FPGA 的重配置，这个过程通常耗时较长（从毫秒级到秒级），中断服务，对于需要快速响应或持续服务的系统是不可接受的 [14]。

为了克服静态 FPGA 设计的局限性，动态部分重构（Dynamic Partial Reconfiguration, DPR）技术应运而生 [15, 16]。DPR 允许在 FPGA 运行时，仅对其内部指定的一部分区域（称为可重构分区，Reconfigurable Partition, RP）进行重新编程，加载新的硬件逻辑（称为可重构模块，Reconfigurable Module, RM），而 FPGA 的其他部分（静态区域和其他 RP）可以保持正常工作，不被中断 [17]。这项技术极大地增强了 FPGA 的灵活性和资源利用率。通过 DPR，可以在有限的 FPGA 资源上分时复用不同的加速功能，或者根据输入数据的特性（如稀疏矩阵的格式或密度）动态加载最优的处理模块，从而实现“按需定制”的硬件加速。

将 DPR 技术应用于矩阵乘法加速，特别是涉及到稀疏矩阵的复杂场景，具有重要的研究价值和应用前景。稀疏矩阵在现实世界中无处不在（如社交网络关系、物理模拟、自然语言处理中的词袋模型等），其存储和计算方式与稠密矩阵截然不同，且存在多种存储格式，不同格式适用于不同的计算场景和优化策略 [18]。一个能够动态切换稀疏矩阵处理逻辑（如不同格式的解压缩、压缩、乘法核心）的 FPGA 加速系统，将能更高效地适应多样化的应用需求，提高硬件资源的利用率和系统的整体性能。

本设计正是基于上述背景，旨在探索和实现一个基于 FPGA 动态部分重构技术的矩阵乘法加速系统。系统以 Xilinx Kria KV260 视觉 AI 入门套件为硬件平台，该平台搭载了 Zynq UltraScale+ MPSoC，集成了强大的 ARM 处理核心和 FPGA 可编程逻辑，构成了典型的异构计算系统 [19]。利用 MPSoC 的 FPGA 部分实现硬件加速，利用 ARM 核心运行 Linux 操作系统进行任务调度、资源管理和与 FPGA 的交互。通过在 FPGA 上划分多个可重构分区，并利用高层次综合（HLS）语言（如 C/C++）设计可重构的矩阵运算模块（包括稀疏矩阵解压、稠密矩阵乘法、稀疏矩阵压缩等），实现了这些模块在运行时的动态加载和切换。

系统通过标准的 AXI 接口协议实现 FPGA 内部模块间以及 FPGA 与处理器系统（PS）端内存的数据交互。结合 Xilinx Runtime (XRT) 库，在运行于 CPU 上的应用程序可以方便地控制 FPGA 上的可重构模块，实现异构协同计算。

第二节　国内外研究现状

近年来，利用 FPGA 加速矩阵运算已成为研究热点。

一、静态 FPGA 加速

许多研究集中于优化特定类型的矩阵乘法。对于稠密矩阵乘法，研究者们探索了各种并行计算架构（如脉动阵列）、分块策略、片上存储优化以及利用 HLS 简化设计流程的方法，以期达到接近理论峰值的性能 [20, 21]。对于稀疏矩阵乘法（SpMV, SpMM），挑战主要在于处理不规则的内存访问和计算模式。研究工作通常针对特定的稀疏格式（如 CSR, CSC, COO, ELL）设计专门的硬件结构，利用流水线、数据预取、负载均衡等技术来缓解访存瓶颈和提高计算单元利用率 [22, 23, 24]。一些工作还尝试设计支持多种稀疏格式的“通用”稀疏矩阵乘法器，但这往往以牺牲部分性能或增加资源消耗为代价 [25]。

二、动态重构技术应用

DPR 技术本身已经相对成熟，并在通信、自适应滤波、软件定义无线电等领域有所应用 [26, 27]。在计算加速领域，DPR 被用于根据需要加载不同的加密算法、图像处理算子或数据压缩算法 [28, 29]。将 DPR 用于矩阵运算的研究相对较少，但已有初步探索。例如，有研究尝试根据矩阵的尺寸或特性动态调整 FPGA 上矩阵乘法器的并行度或数据位宽 [30]。也有工作利用 DPR 在不同类型的数值计算核心（如浮点加法器、乘法器）之间切换。

三、异构计算与运行时管理

随着 SoC FPGA（如 Xilinx Zynq 系列）的普及，基于 FPGA 的异构计算系统成为主流。研究者们关注如何高效地在处理器和 FPGA 之间划分任务、传输数据以及管理 FPGA 上的硬件资源 [31]。Xilinx PYNQ 框架和 XRT 运行时库等工具的出现，极大地简化了在高级操作系统（如 Linux）层面调用 FPGA 加速器的开发流程 [32, 33]。

四、现有研究的局限性

尽管已有大量关于 FPGA 矩阵加速和 DPR 技术的研究，但将两者深度结合，特别是针对包含多种稀疏/稠密矩阵操作、需要动态适应不同稀疏格式的复杂计算流，进行系统性设计和实现的研究尚不多见。现有工作要么侧重于静态优化单一类型的矩阵运算，要么 DPR 的应用场景相对简单，未能充分发挥 DPR 在处理复杂、多变矩阵计算任务流中的潜力。此外，如何在 HLS 设计流程中高效地集成 DPR，以及如何通过 XRT 等运行时环境有效管理多个可重构分区的动态加载和任务调度，也是需要进一步探索的问题。

第三节　本设计的主要工作与创新点

本毕业设计针对现有研究的不足，着眼于设计并实现一个动态可重构的 FPGA 矩阵乘法加速系统，旨在提供一个既高效又灵活的解决方案，以适应现代应用中复杂多变的矩阵计算需求。具体工作和创新点如下：

基于 DPR 的灵活矩阵运算硬件架构设计 在 Xilinx KV260 平台的 FPGA 上，设计并实现了包含三个独立可重构分区（RP）的硬件架构。这种多 RP 架构允许多个不同的矩阵运算模块（RM）共存或被快速替换，为实现复杂的计算流水线或并行处理不同任务提供了硬件基础。

面向多类型矩阵运算的可重构模块开发 使用 Vitis HLS 工具，以 C/C++ 语言开发了一系列针对不同矩阵运算任务的 RM，包括：

- 稀疏矩阵解压缩（将特定稀疏格式转换为稠密格式）
- 稠密矩阵乘法（支持结果输出为稠密或稀疏格式）
- 稀疏矩阵压缩（将稠密格式转换为特定稀疏格式）
- （隐含支持）稀疏矩阵格式转换（可通过解压 + 压缩 RM 组合实现）
- （隐含支持）稀疏矩阵乘法（可通过解压 + 稠密乘法或解压 + 稠密乘法 + 压缩等 RM 组合/序列实现，或未来直接开发专用稀疏乘法 RM）

这些模块被设计为可动态加载到 RP 中，使得系统能够根据具体任务需求，实时配置最优的硬件加速逻辑，特别是能够灵活切换对不同稀疏矩阵格式的处理能力。

标准化接口与异构系统集成 计的 RM 均采用标准的 AXI 接口协议。AXI-MM 接口用于高效访问 KV260 的共享 DDR 内存，实现大规模数据的吞吐；AXI-Stream 接口用于连接不同的 RM，支持在 FPGA 内部构建数据流驱动的计

算流水线。系统运行在 KV260 的 ARM 处理器上的 Ubuntu Linux 操作系统，通过 Xilinx Runtime (XRT) 库，实现了上层软件对 FPGA 硬件资源（包括 DPR 操作和 RM 任务执行）的统一管理和调用，构建了一个完整的异构加速计算平台。

动态适应性与应用场景验证 重点验证了系统的动态可重构特性。通过编程示例，展示了系统如何根据指令，在运行时动态地将不同的 RM 加载到 RP 中，以执行不同的加速计算任务序列（例如，先解压稀疏矩阵 A，然后与稠密矩阵 B 相乘，最后将结果压缩为稀疏格式 C）。这证明了系统在适应不同计算流程、不同数据格式方面的灵活性优势，尤其适用于需要处理多种稀疏表示或在稀疏/稠密计算间切换的应用场景。

本设计的创新之处在于：系统性地将动态部分重构技术应用于涵盖多种（稀疏、稠密、转换）矩阵运算的 FPGA 加速场景，并构建了一个包含多 RP、HLS 设计的 RM、标准 AXI 接口、以及基于 Linux+XRT 的软硬件协同控制的完整异构系统。这为应对未来更复杂、更动态的计算挑战提供了一种有潜力的技术途径。

第四节　论文结构安排

本论文共分为七章，结构安排如下：**第一章：绪论**主要介绍研究背景、意义、国内外研究现状、本设计的主要工作与创新点以及论文的结构安排。**第二章：相关技术概述**详细介绍本设计所涉及的关键技术，包括矩阵运算基础、FPGA 技术原理、高层次综合（HLS）、动态部分重构（DPR）技术、AXI 总线协议、异构计算系统以及 Xilinx KV260 平台和 XRT 运行时环境。**第三章：系统总体设计**阐述动态可重构矩阵乘法加速系统的整体架构，包括硬件系统设计（FPGA 分区规划、静态与动态区域划分、接口设计）和软件系统设计（操作系统层面、XRT 应用层面、任务调度逻辑）。**第四章：可重构模块硬件实现**详细介绍使用 Vitis HLS 设计和实现各个可重构矩阵运算模块（稀疏解压、稠密乘法、稀疏压缩等）的过程，包括算法分析、HLS 优化（并行、流水线）、接口实现以及综合与实现结果。**第五章：软件系统实现与集成**描述在 KV260 的 ARM 处理器上配置 Linux 环境、安装 XRT、编写主机端应用程序以控制 DPR 流程（加载/卸载 RM）、管理数据传输以及调用 FPGA 执行加速任务的具体实现方法。**第六章：实验结果与分析**对实现的系统进行测试和评估。包括功能验证、DPR 操作时间测量、各矩阵运算任务在 FPGA 上的加速性能测试（与纯 CPU 实现对比），以及系统灵活性和资源利用

率的分析。**第七章：总结与展望**总结本论文完成的主要工作和取得的研究成果，分析存在的不足，并对未来可以进一步研究的方向进行展望。

第二章　相关技术概述

本章旨在详细介绍本工作所涉及的核心技术。这些技术涵盖了从基础算法理论到硬件平台、设计方法学、接口协议以及软件运行时环境等多个层面。深入理解这些技术对于后续章节中系统设计、实现与评估的阐述至关重要。本章将依次介绍矩阵运算基础、FPGA 技术原理、高层次综合（HLS）、动态部分重构（DPR/DFX）技术、AXI 总线协议、异构计算系统、Xilinx Kria KV260 平台特性以及 Xilinx Runtime（XRT）环境。

第一节　矩阵运算基础

矩阵作为一种重要的数据结构，在科学计算、图像处理、机器学习、组合优化等诸多领域扮演着核心角色。矩阵运算，尤其是矩阵乘法，通常是这些应用中计算量最为密集的部分之一。根据矩阵中非零元素的分布，矩阵可分为稠密矩阵和稀疏矩阵。稠密矩阵的元素大多为非零值，而稀疏矩阵则包含大量的零元素。针对稀疏矩阵的特性，发展出了多种压缩存储格式，如坐标列表（COO）、压缩稀疏行（CSR）、压缩稀疏列（CSC）等，以节省存储空间并优化计算效率。本项目涉及的核心计算任务包括稀疏矩阵与稠密矩阵之间的相互转换（解压与压缩）、稀疏矩阵格式之间的转换，以及稠密矩阵乘法和稀疏矩阵乘法。这些运算，特别是涉及大规模矩阵时，对计算性能提出了极高要求，从而驱动了对硬件加速方案的探索。

第二节　FPGA 技术原理

现场可编程门阵列（FPGA）是一种半定制电路，其硬件结构可以在制造完成后由用户根据需求进行配置。FPGA 内部主要由可配置逻辑块（CLB）、输入输出块（IOB）、可编程布线资源以及嵌入式存储器（如 BRAM）、数字信号处理单元（DSP Slice）等构成。用户通过加载特定的配置文件（比特流）来定义这些单元的功能及其互连方式，从而实现所需的数字逻辑电路。与通用处理器相比，FPGA 能够实现高度的并行计算和深流水线操作，充分利用硬件资源，从而在特定计算密集型任务上展现出显著的性能优势和能效比。其可重构性也为算法的

迭代升级和功能调整提供了灵活性。

第三节　高层次综合

高层次综合（High-Level Synthesis, HLS）是一种设计方法，它允许开发者使用诸如 C、C++ 或 SystemC 等高层次语言来描述硬件行为，然后通过 HLS 工具将其自动转换为低层次的硬件描述语言（HDL），如 Verilog 或 VHDL。HLS 的出现极大地提高了 FPGA 的设计效率，缩短了开发周期，并使得不熟悉底层 HDL 的软件工程师也能够参与到 FPGA 开发中。在本项目中，可重构模块采用 Vitis HLS 进行编写，通过利用 HLS 提供的并行化、流水线化等优化指令（pragmas），能够有效地将算法映射到 FPGA 的硬件资源上，实现高效的硬件加速器设计。

第四节　动态部分重构技术

动态部分重构（Dynamic Partial Reconfiguration, DPR），在 Xilinx 平台中也称为 Dynamic Function eXchange (DFX)，是一项先进的 FPGA 技术。它允许在 FPGA 正常运行期间，对其内部的特定区域（即可重构分区, Reconfigurable Partition, RP）进行动态地、部分地重新编程，而 FPGA 的其他部分保持原有功能并继续工作。这项技术为系统带来了极大的灵活性，使得 FPGA 能够根据应用需求实时切换不同的硬件加速模块（可重构模块, Reconfigurable Module, RM），或者在线更新硬件功能，而无需中断整个系统的运行。在本项目中，通过在 FPGA 上成功布局 3 个可重构分区，并部署不同的矩阵运算模块，实现了根据计算任务动态加载和切换功能，例如在不同稀疏矩阵格式处理模块间进行切换，以适应多样化的加速场景。

第五节　AXI 总线协议

高级可扩展接口（Advanced eXtensible Interface, AXI）是 ARM 公司提出的一种高性能、高带宽、低延迟的片上总线协议，已成为业界标准，广泛应用于 SoC（System-on-Chip）设计中。AXI 协议定义了主设备（Master）和从设备（Slave）之间的通信规范，支持多种通信模式。本项目中主要使用了两种 AXI 接口：AXI Memory Mapped (AXIMM) 协议和 AXI Stream (AXIS) 协议。AXIMM 协议用于可重构分区对 KV260 的内存进行直接访问（DMA），实现高效的数据读写；AXIS

协议则用于可重构模块之间的互联，支持高速、连续的数据流传输，非常适合流水线式的数据处理架构。

第六节　异构计算系统

异构计算系统是指在一个系统中集成多种不同类型计算单元（如 CPU、GPU、FPGA、DSP 等）的计算平台。这种架构旨在结合不同处理单元的优势，例如 CPU 擅长处理复杂的控制流和非结构化任务，而 FPGA 则擅长执行大规模并行计算和定制化的数据处理。通过合理的任务划分和协同工作，异构计算系统能够实现比单一类型处理器更高的性能和能效。本项目基于 Xilinx KV260 MPSoC 构建了一个异构计算系统，其中 ARM Cortex-A53 CPU 负责运行操作系统（Ubuntu Linux）和上层应用程序，并调度 FPGA 上的硬件加速模块执行计算密集型的矩阵运算任务。

第七节　Xilinx Kria KV260 平台特性

Xilinx Kria KV260 视觉 AI 入门套件是基于 Kria K26 系统级模块（SOM）的开发平台。K26 SOM 是一款多处理器系统芯片（MPSoC），集成了四核 ARM Cortex-A53 处理系统（PS）和拥有丰富可编程逻辑（PL）资源的 FPGA。KV260 平台专为边缘 AI 和视觉应用设计，但其强大的异构处理能力和灵活的接口使其也适用于其他加速计算任务。它提供了 DDR4 内存、多种外设接口以及对 Xilinx 开发工具链的良好支持。其 MPSoC 架构天然支持 PS 与 PL 之间的高效协同，是实现本项目中 CPU 控制、FPGA 加速的理想硬件基础。FPGA 部分支持动态部分重构，为实现灵活可变的加速器提供了硬件保障。

第八节　Xilinx 运行时环境

Xilinx Runtime (XRT) 是一个开源的、标准化的软件平台，旨在简化主机 CPU 与 Xilinx FPGA 加速器之间的交互。XRT 包括用户空间库、API 以及内核驱动程序，它为应用程序提供了一个统一的接口来管理和控制 FPGA 上的加速内核，无论这些内核是部署在 Alveo 数据中心加速卡还是像 KV260 这样的嵌入式 SoC 平台上。XRT 负责处理诸如加载比特流（包括部分重构的比特流）、分配和迁移内存缓冲区、调度内核执行以及收集性能数据等任务。在本项目中，部署在 KV260

的 CPU 上的 Ubuntu Linux 操作系统中，应用程序通过 XRT 提供的 API 来调用和管理 FPGA 上的动态可重构矩阵运算模块，实现了高效的异构加速计算流程。

第三章　系统总体设计

在深入探讨具体的硬件实现和软件编程细节之前，本章将阐述动态可重构矩阵乘法加速系统的整体架构。系统设计遵循软硬件协同设计（Hardware/Software Co-design）的理念，旨在充分利用 Xilinx Kria KV260 平台的异构计算能力，结合动态部分重构（Dynamic Partial Reconfiguration, DFX）技术，实现一个既高性能又具备高度灵活性的矩阵运算加速解决方案。本章将分别从硬件系统架构和软件系统架构两个层面进行阐述，并明确两者之间的交互方式与核心接口设计，为后续章节的详细实现奠定基础。

第一节　硬件系统架构

硬件系统的核心是 Xilinx Kria KV260 多处理器系统芯片（MPSoC），其集成了可编程逻辑（Programmable Logic, PL）即 FPGA，以及处理系统（Processing System, PS）即 ARM CPU。本设计的 FPGA 部分充分利用了其并行处理能力和动态可重构特性。

在 FPGA 的整体规划上，我们将其划分为静态区域（Static Region）和动态区域（Dynamic Regions），即所谓的可重构分区（Reconfigurable Partitions, RPs）。静态区域承载了系统运行所必需的基础逻辑，例如与 PS 的接口控制器、时钟管理、中断管理以及用于加载动态模块的重构控制器（如 ICAP）。本设计成功实现了三个独立的可重构分区，这三个分区为动态加载不同的计算模块提供了物理基础。每个可重构分区均设计为能够容纳一个可重构模块（Reconfigurable Module, RM）。具体部署的 RMs 包括稀疏矩阵解压模块、采用脉动阵列结合分块策略的稠密矩阵乘法模块以及稀疏矩阵压缩模块。这些模块均采用 Vitis 高层次综合（HLS）语言编写，通过精心设计的 pragma 指令，实现了高度的并行计算和流水线操作，从而最大化地利用 FPGA 的硬件资源，提升运算效率。

接口设计是确保数据高效流转和模块协同工作的关键。在可重构分区与 KV260 片上内存的交互方面，采用了 AXI Memory Mapped (AXIMM) 主接口协议。这使得每个动态加载的 RM 都能够直接、高效地对系统主存进行读写操作，为大规模矩阵数据的传输提供了高带宽通道。同时，为了实现可重构模块之间的数据流传递和协同处理，例如将解压后的数据直接送入乘法模块，我们设计了

基于 AXI Stream (AXIS) 协议的互联接口。这种流式接口非常适合于流水线式的处理流程，能够有效减少数据在模块间传递的延迟。静态区域与动态区域之间，以及动态区域与 PS 之间的控制与状态信号交互，则通过标准的 AXI Lite 接口或 GPIO 实现。

第二节　软件系统架构

软件系统架构是建立在 KV260 的 ARM 处理器之上，负责管理硬件资源、调度计算任务以及提供用户交互接口。整个软件栈可以划分为操作系统层面、Xilinx 运行时（XRT）应用层面以及顶层的任务调度逻辑。

在操作系统层面，KV260 的 CPU 上部署了 Ubuntu Linux 操作系统。Linux 系统为上层应用提供了稳定的运行环境和丰富的系统服务，包括内存管理、进程调度以及设备驱动程序等。FPGA 作为一种可编程设备，其驱动和管理由 Linux 内核模块以及 Xilinx 提供的运行时库共同完成。

Xilinx 运行时（XRT）是连接用户空间应用程序与 FPGA 加速硬件之间的关键桥梁。XRT 提供了一套标准的 API，应用程序可以通过这些 API 来管理 FPGA 设备、分配和迁移数据缓冲区、加载 FPGA 比特流（包括用于动态重构的部分比特流）以及控制加速内核的执行。在本系统中，我们编写的 C/C++ 应用程序利用 XRT 提供的接口，实现了对 FPGA 上动态加载的矩阵运算模块的调用和管理，从而实现异构加速计算。

任务调度逻辑是本系统灵活性的核心体现。根据用户请求的加速计算任务类型，该逻辑负责决策在 FPGA 的三个可重构分区上分别加载哪些可重构模块。例如，若执行完整的稀疏矩阵乘法（结果为稀疏矩阵），调度逻辑可能会依次或并行地在 RPs 中部署稀疏矩阵解压模块、稠密矩阵乘法模块和稀疏矩阵压缩模块。若仅进行稀疏矩阵格式转换，则会加载相应的转换模块。这一调度过程包括了动态加载所需的部分比特流到指定 RP，并通过 XRT 配置模块参数、启动运算、以及监控运算状态。由于所有计算模块均支持动态重构，系统能够根据不同的稀疏矩阵格式或计算需求（例如，结果是稠密矩阵还是稀疏矩阵）灵活切换 FPGA 上的功能模块，以适应各种加速场景。这种动态性是本系统区别于传统静态 FPGA 加速方案的关键优势。

第三节　软硬件协同与系统工作流程

本系统的核心在于软硬件的紧密协同。典型的加速计算任务流程始于用户应用程序通过 XRT API 发起请求。CPU 上的任务调度逻辑接收此请求，分析任务需求（如操作类型、矩阵格式、数据位置等）。随后，调度逻辑判断当前 FPGA 各可重构分区上的模块是否满足需求。若不满足，则通过 XRT 触发动态部分重构流程，将合适的 RM 加载到相应的 RP 中。数据准备阶段，CPU 通过 XRT 将待处理的矩阵数据从主存传输到 FPGA 可访问的内存区域，或直接映射主存供 FPGA 的 AXIMM 接口访问。一旦模块加载完毕且数据就绪，XRT 便会启动 FPGA 上的计算内核。FPGA 模块通过 AXIMM 接口读取数据，执行计算（可能涉及多个 RM 通过 AXIS 接口流水线作业），并将结果写回内存。计算完成后，FPGA 通过中断或轮询方式通知 CPU，CPU 再通过 XRT 将结果数据传回用户应用程序，或通知应用程序结果已就绪。整个过程充分利用了 CPU 的控制能力和 FPGA 的并行计算能力，并通过动态重构技术实现了高度的灵活性和资源利用率。

第四章　可重构模块硬件实现

本章详细阐述构成动态可重构矩阵乘法加速系统核心计算能力的可重构模块（Reconfigurable Modules, RMs）的设计与实现过程。这些模块是专门为加载到第三章定义的三个可重构分区（RPs）中而设计的。我们将采用 Xilinx Vitis 高层次综合（HLS）工具，利用 C/C++ 语言进行算法描述和硬件优化。对于每个关键的 RM（稀疏矩阵解压、稠密矩阵乘法、稀疏矩阵压缩），本章将覆盖其核心算法分析、HLS 实现策略、接口设计、关键优化技术以及最终的综合与实现结果（基于目标平台 Xilinx Kria KV260 的 Zynq UltraScale+ MPSoC PL 部分的资源特性）。

第一节　Vitis HLS 设计方法论概述

itis HLS 作为一种高层次综合工具，允许设计者使用 C、C++ 或 OpenCL C 等高级语言描述硬件行为，然后将其自动转换为寄存器传输级（RTL）代码。这种方法显著提高了设计效率，使得复杂的算法能够更快地在 FPGA 上实现。在本系统中，所有可重构模块均采用 C++ 进行开发。核心设计策略是首先实现算法的功能正确性，然后通过迭代地应用 HLS 优化指令（pragmas）来提升性能和资源利用率。关键的优化指令包括用于实现指令级并行的 `PIPELINE` 指令，用于循环展开以实现数据级并行的 `UNROLL` 指令，用于优化存储器访问的 `ARRAY_PARTITION` 指令，以及用于定义模块接口（如 AXI4-Stream, AXI4-Lite, AXI4-Master）的 `INTERFACE` 指令。设计目标是为每个 RM 生成高效的、能够充分利用 FPGA 并行性和流水线性质的硬件实现。

第二节　稀疏矩阵解压/压缩模块

稀疏矩阵解压模块负责将以特定稀疏格式（例如 CSR, COO 等）存储的矩阵转换为稠密矩阵格式。其核心算法依赖于稀疏格式的定义。以 CSR（Compressed Sparse Row）格式为例，输入数据通常包括值数组（values）、列索引数组（column indices）和行指针数组（row pointers）。

稀疏矩阵压缩模块的功能与解压模块相反，它将稠密矩阵转换为指定的稀

疏格式。

一、算法分析与 HLS 实现

稀疏矩阵解压模块的算法逻辑是遍历行指针数组，确定每行非零元素的数量和位置，然后根据列索引数组和值数组将这些非零元素填充到稠密矩阵的相应位置，其余位置则填充零。在 HLS 实现中，通常会涉及嵌套循环：外层循环遍历行，内层循环遍历该行内的非零元素。为了高效访问输入稀疏数据（通常存储在主存中），模块通过 AXIMM 主接口读取。输出的稠密矩阵同样通过 AXIMM 主接口写回主存，或者通过 AXIS 接口流式传输给下一个处理模块。

稀疏矩阵压缩模块模块首先遍历输入的稠密矩阵，识别所有非零元素，并记录它们的值和索引。随后，根据目标稀疏格式（如 CSR）的要求，将这些信息组织成相应的数组结构。例如，生成 CSR 格式需要统计每行的非零元素个数以构建行指针数组，同时记录非零元素的值和列索引。

二、综合与实现结果

在针对 KV260 平台进行综合后，模块在消耗合理的 LUT、FF、BRAM 资源的前提下，旨在达到较高的工作时钟频率。具体的资源消耗和性能数据将通过 Vitis HLS 综合报告和 Vivado 实现报告获得，并根据不同的稀疏度与矩阵维度进行评估。

第三节　稠密矩阵乘法模块

稠密矩阵乘法模块是本加速系统的核心计算单元之一，其设计采用了脉动阵列（Systolic Array）结构，并结合分块（Tiling）策略以处理大规模矩阵。

一、算法分析与 HLS 实现

脉动阵列因其规整的数据流和高度的并行性，非常适合在 FPGA 上实现矩阵乘法。算法将输入矩阵 A 和 B 分割成小块（tiles），然后将这些小块数据送入脉动阵列进行计算。每个处理单元（PE）在脉动阵列中执行乘加运算。分块策略有助于管理片上存储资源（如 BRAM），使得模块可以处理远大于片上存储容量的矩阵。HLS 实现中，需要精心设计数据加载、PE 阵列的计算逻辑以及结果写回的控制流程。

二、HLS 优化策略与接口设计

针对脉动阵列的 PEs，其内部的乘加操作循环体应用 `PIPELINE` 指令是标准做法。为了实现 PE 之间的数据并行流动，相关的循环通常会被完全 `UNROLL`。用于缓存输入矩阵分块的片上存储器（BRAMs）会使用 `ARRAY_PARTITION` 指令进行分区，以提供足够的并行读写带宽。接口方面，输入矩阵数据（或其分块）通常通过 AXIS 从接口输入，这便于从前级模块（如解压模块）或 PS 端高效接收数据。计算结果（稠密矩阵或其分块）通过 AXIS 主接口输出，可以流向后级模块（如压缩模块）或 PS 端。对于直接从主存加载分块或写回分块的场景，也会配置 AXIMM 主接口。

三、综合与实现结果

脉动阵列的大小（即 PE 数量）和分块大小是关键的设计参数，它们直接影响资源消耗和性能。目标是在 KV260 的资源约束下最大化 PE 数量和工作频率。综合结果将展示 LUT、FF、DSP（用于乘法运算）和 BRAM 的使用情况，以及预估的吞吐率。

第四节　模块间的协同与动态特性

以上设计的可重构模块均具备标准的 AXI 接口（AXIMM 用于内存访问，AXIS 用于流式数据传输），这使得它们不仅可以独立工作，也可以在 FPGA 的三个可重构分区中灵活组合，通过 AXIS 接口形成高效的数据处理流水线。例如，稀疏矩阵乘法任务可以通过动态加载稀疏解压 RM、稠密乘法 RM 和稀疏压缩 RM，并使它们通过 AXIS 接口依次串联工作。Vitis HLS 生成的模块是参数化的，能够处理不同维度和稀疏格式的矩阵（通过重新配置或在模块内部设计一定的灵活性），结合动态部分重构技术，系统可以根据具体任务需求，在运行时切换 FPGA 上的硬件功能，充分体现了设计的灵活性和高效性。每个 RM 都作为独立的编译单元生成部分比特流，为第三章所述的动态重构系统提供了基础。

请注意，本章草案中提及的“综合与实现结果”部分，您需要在完成实际的 HLS 综合和 Vivado 实现后，用具体的资源占用数据（LUTs, FFs, DSPs, BRAMs）和时钟频率来填充。

第五章　软件系统实现与集成

在前两章分别完成系统总体架构设计和可重构硬件模块实现后，本章将聚焦于运行在 Xilinx Kria KV260 平台处理器系统（PS）上的软件系统实现与集成。这包括配置嵌入式 Linux 操作环境、安装和配置 Xilinx Runtime (XRT) 库，以及最关键的——编写主机端 C++ 应用程序。该应用程序负责解析用户任务、通过 XRT 动态管理 FPGA 上的可重构分区（RPs）和可重构模块（RMs）、高效地传输数据，并精确地调用和协调 FPGA 上的硬件加速任务。本章旨在详述将硬件能力转化为可用、灵活的加速服务的软件实现路径。

第一节　嵌入式 Linux 环境配置

Kria KV260 Vision AI 入门套件的处理器系统端运行嵌入式 Linux。本系统采用 Xilinx 官方提供的 PetaLinux 构建的 Linux 发行版，该发行版针对 Zynq UltraScale+ MPSoC 进行了优化，并内建了对 FPGA 可编程逻辑（PL）以及动态功能交换（DFX，即动态部分重构）的支持。环境配置的首要步骤是获取或定制包含必要内核驱动和设备树（Device Tree）配置的 Linux 镜像。设备树中需正确声明 FPGA 管理器、可重构分区以及相关的时钟和接口资源，确保操作系统能够识别并管理 PL 端的硬件。启动 KV260 并加载 Linux 系统后，会进行基础的系统配置，例如网络连接、用户管理，并安装必要的开发工具链，如 GCC/G++ 编译器、Make 等，为后续 XRT 安装和主机应用程序编译提供支持。截至 2025 年，PetaLinux 工具和 Kria 平台的 Linux 镜像已对 DFX 提供了成熟的支持。

第二节　主机应用程序设计与架构

主机应用程序采用 C++ 语言编写，旨在提供一个用户友好的接口来调用 FPGA 实现的各种矩阵运算任务。其核心架构围绕任务解析、FPGA 资源管理（包括 DPR）、数据准备与传输、以及内核执行控制这几个关键功能模块展开。应用程序首先会初始化 XRT 环境，打开与目标 FPGA 设备的连接。随后，它会进入一个主循环或响应用户输入，接收具体的计算任务请求，例如“执行稀疏矩阵 A 与稀疏矩阵 B 的乘法，结果以稀疏格式输出”。

任务解析模块负责将用户请求分解为一系列 FPGA 操作。例如，上述稀疏矩阵乘法任务可能被分解为：加载稀疏解压 RM 到 RP1，加载另一个稀疏解压 RM（或复用 RP1）到 RP2（如果并行处理两个输入矩阵），加载稠密矩阵乘法 RM 到 RP2 或 RP3，以及加载稀疏压缩 RM 到 RP3。如果 RP 数量有限（本设计为 3 个），则可能需要串行化部分操作，并将中间结果暂存到 DDR 内存中。

第六章　实验结果与分析

本章旨在对前述章节设计并实现的动态可重构 FPGA 矩阵乘法加速系统进行全面的测试与评估。我们将通过一系列实验来验证系统的功能正确性、测量动态重构的关键性能指标、评估核心矩阵运算任务的加速效果（与 CPU 基准对比）、分析系统动态灵活性带来的优势，并量化硬件资源的利用率。本章的目标是提供实验证据，证明所提出设计的有效性和实用性。

第一节　实验环境与配置

实验硬件平台为 Xilinx Kria KV260 Vision AI 入门套件，其核心是 Zynq UltraScale+ MPSoC。可编程逻辑（PL）部分根据第四章所述的可重构模块设计，在综合实现后，各模块的工作时钟频率设定为 250MHz。处理器系统（PS）端的 ARM Cortex-A53 四核处理器运行频率为 1.2GHz。软件环境方面，KV260 上部署了基于 PetaLinux 2023.2 构建的 Ubuntu 22.04 LTS 操作系统。Xilinx Runtime (XRT) 版本为 2.16，主机应用程序采用 C++17 编写，并使用 GCC 11.4.0 进行编译。CPU 基准性能测试在同一 KV260 的 ARM 处理器上进行（单线程的朴素矩阵乘法）。

第二节　系统功能验证

功能验证是确保系统按预期工作的首要步骤。我们针对第五章中描述的所有加速计算任务进行了测试：稀疏矩阵解压为稠密矩阵、稠密矩阵压缩为稀疏矩阵、稀疏矩阵格式转换（例如，从假设的 COO 格式 RM 到 CSR 格式 RM 的转换流程）、稠密矩阵乘法（结果分别为稠密和稀疏），以及稀疏矩阵乘法（结果分别为稠密和稀疏）。对于每项任务和不同规模的测试数据，FPGA 加速器的计算结果均与在 CPU 上使用 Eigen 库或自定义 C++ 实现的“黄金参考”结果进行比对。实验结果表明，所有 FPGA 加速任务的输出均与 CPU 参考结果一致（在单精度浮点数允许的误差范围内），从而验证了整个系统（包括 HLS 模块设计、AXI 接口、XRT 调用、DPR 流程及主机应用程序逻辑）的功能正确性。

第三节　GEMM 加速性能分析

我们对核心矩阵运算任务在 FPGA 上的加速性能进行了评估，并与纯 CPU 执行时间进行了对比。性能指标主要关注端到端执行时间，包括数据从主机 PS DDR 传输到 PL 端 DDR、FPGA 内核执行以及结果数据从 PL 端 DDR 传回 PS DDR 的时间。

对于稠密矩阵乘法 (GEMM)，我们测试了 512x512 单精度浮点稠密矩阵乘法，CPU（执行时间约为 C1 毫秒，而 FPGA 上的稠密矩阵乘法 RM 总共耗时 F1 毫秒（包括数据传输时间和内核执行时间），实现了约 S1 倍的加速。

第四节　系统灵活性与适应性分析

本系统设计的核心优势之一在于其动态可重构性带来的灵活性。实验中，我们能够通过主机应用程序的控制，在 2 至 4 毫秒的时间内更换 FPGA 上特定可重构分区的功能模块。这使得系统能够：

1. 适应不同的稀疏矩阵格式：通过加载针对特定稀疏格式（如 CSR, CSC, COO 等）优化的解压或压缩 RM，系统无需重新综合整个 FPGA 设计即可处理多种数据输入。例如，从处理 CSR 格式切换到处理 COO 格式，仅需重配置相应的 RM。
2. 构建任务定制的加速流水线：根据具体的计算需求，动态组合不同的 RMs。例如，若仅需稀疏矩阵解压，则只加载解压 RM；若需完整的 SpGEMM，则加载解压、乘法、压缩三个 RMs。这种按需加载的方式避免了在 FPGA 上静态集成所有可能模块而导致的资源浪费。
3. 优化资源利用：在不同计算阶段，可以将不再需要的 RM 替换为后续阶段所需的 RM，从而使得有限的 FPGA 资源得到更高效的复用。例如，对于一个大规模稀疏矩阵乘法，如果不能同时容纳两个解压模块和一个乘法模块，可以先解压一个矩阵，然后用其解压模块的 RP 重配置为第二个解压模块或直接用于乘法模块（若解压 RM 输出直接到 DDR）。

这种灵活性使得系统能够以接近定制硬件的性能，应对多样化的计算场景，而无需为每种场景设计独立的静态 FPGA 加速器。

第五节　讨论与分析总结

实验结果综合表明，所设计的动态可重构 FPGA 矩阵乘法加速系统成功实现了预期的功能，并在 GEMM 运算任务上展现了相对于 ARM CPU 的显著加速效果。稠密矩阵乘法的脉动阵列设计和稀疏处理流水线的构建，有效地利用了 FPGA 的并行计算能力。动态部分重构机制虽然引入了一定的时间开销，但其赋予系统的灵活性和适应性，使其能够高效应对多种计算需求和数据格式，这对于需要处理多样化任务的场景具有重要价值。

第七章　总结

本论文围绕在 Xilinx Kria KV260 多处理器系统芯片（MPSoC）上构建一个高性能、高灵活性的动态可重构矩阵乘法加速系统展开研究与实现。面对传统固定功能加速器在处理多样化矩阵运算任务及适应不同数据格式方面的局限性，本文提出并成功实现了一个基于动态部分重构（DFX）技术的异构计算解决方案。

首先，在系统总体设计层面（第三章），我们规划了基于 KV260 的软硬件协同架构。硬件上，在 FPGA 的可编程逻辑（PL）部分划分了静态区域和三个独立的可重构分区（RPs）。静态区域负责基础平台支持，而 RPs 则用于动态加载不同的计算模块。软件上，在处理器系统（PS）端的 ARM 处理器上部署了 Ubuntu Linux 操作系统，并利用 Xilinx Runtime (XRT) 作为主机应用程序与 FPGA 硬件交互的桥梁。

其次，在可重构模块硬件实现层面（第四章），我们采用 Vitis 高层次综合（HLS）语言，设计并优化了一系列核心矩阵运算的可重构模块（RMs）。这些模块包括稀疏矩阵解压模块、采用脉动阵列结合分块策略的稠密矩阵乘法模块以及稀疏矩阵压缩模块。通过精心设计的并行与流水线优化，这些 RM 能够高效利用 FPGA 资源。模块间通过 AXI4-Stream (AXIS) 协议实现高速数据互联，并通过 AXI4 Memory Mapped (AXIMM) 协议直接访问 KV260 的系统内存。

再次，在软件系统实现与集成层面（第五章），我们详细阐述了在 KV260 的 ARM 处理器上配置 Linux 环境、安装 XRT，并重点介绍了主机端 C++ 应用程序的设计。该应用程序负责解析用户请求，通过 XRT 动态管理 FPGA 上的 RPs（加载/卸载 RMs），高效组织数据在 PS 与 PL 间的传输，并精确控制 FPGA 加速内核的执行流程，实现了对整个异构加速系统的有效调度。

最后，通过一系列实验（第六章），我们对所构建的系统进行了全面的功能验证、DPR 操作性能测量、核心矩阵运算任务的加速性能评估（与 CPU 基准对比），并分析了系统的灵活性和硬件资源利用率。

参 考 文 献

致　谢

感谢宫磊老师对本工作的指导。

2025 年 5 月