本科毕业设计（论文）开题报告

课题名称：基于 **RISC-V** 存算一体芯片的编译器关键技术研究

学 员 姓 名：	简泽鑫	学　　号：	202102001019
首次任职专业：	无	学历教育专业：	计算机科学与技术（计算机系统）
命 题 学 院：	计算机学院	年　　级：	2021 级
指 导 教 员：	曾坤	职　　称：	副研究员
所 属 单 位：	计算机学院微电子与微处理器研究所		

国防科技大学教育训练部制

一、 课题名称、来源、选题依据

（一） 课题名称

基于 RISC-V 存算一体芯片的编译器关键技术研究

（二） 课题来源

源自老师自选课题

（三） 选题依据

随着人工智能（AI）和大数据应用的迅猛发展，计算需求呈指数级增长，传统的计算架构面临严峻挑战。特别是冯诺依曼架构由于其数据在存储和计算单元之间频繁传输，导致数据移动成为性能瓶颈，限制了系统的整体效率。为了解决这一问题，存算一体（Compute-In-Memory，CIM）架构应运而生，通过将计算功能集成到存储单元中，显著减少数据传输量，从而突破冯诺依曼瓶颈，提升系统性能和能效。

同时，RISC-V 作为一种开放、可扩展的指令集架构（Instruction Set Architecture，ISA），凭借其完全开源、模块化设计和强大的可定制性，已成为构建存算一体芯片的首选。然而，基于 RISC-V 构建的存算芯片具有异构、碎片化的特点，这就要求开发者面向不同存算架构开发多个版本的应用，开发效率低、部署难。因此，如何实现将软件操作（包括计算、数据通信等）和硬件配置（如异构计算单元、存储层次等）解耦，以便 AI 应用开发不再依赖存算 IP 设计，是解决“编程墙”的关键问题。

不仅如此，AI 应用的不规则的发展趋势和存算芯片的异构化、碎片化的现状，使得我们需要探索新的动态编译优化方法，这种优化方法既需要能够充分的考虑到 AI 应用的动态变化的特质，又需要能够充分的挖掘未来存算芯片的架构特征。通过动态编译优化，可以实时调整编译策略，使得生成的代码能够更好地适应硬件的运行环境，提高计算效率和资源利用率。

故，本课题将面向 RISC-V 存算芯片修改 LLVM 编译器，实现对存算指令的支持。通过深入研究 LLVM 编译器架构和工作原理，分析 RISC-V 存算一体芯片的特性，探索如何在 LLVM 中添加对 RISC-V CIM 的支持。这包括但不限于对指令集的扩展、内存模型的适配以及优化策略的调整等，以实现应用算子的自动映射和正确指令流的生成，从而更好地协调计算部件，挖掘芯片内部的计算并行性，为 RISC-V 存算一体芯片提供有力的编译支持。

二、 国内外研究现状及发展趋势

（一） 国内外总体研究情况和发展前景

基于 RISC-V 指令集构建存算一体芯片逐渐成为 AI 加速器主流。一方面，RISC-V 指令集具有高度开放、标准化等优势，适合用领域定制的芯片开发和设计。另一

方面，可以借助 RISC-V 国际社区的力量，通过 RISC-V 扩展标准化的推动和发展促进统一、高效的 AI 编程模型和系统软件支撑框架的形成。因此，谷歌、脸书、微软等巨头，都基于 RISC-V 指令集搭建自有 AI 芯片。2023 年 RISC-V SoC 的市场渗透率达到 2.6%，市场规模 61 亿美元，并正在持续上升。然而，人工智能深度领域定制的趋势导致了 RISC-V 存算一体芯片异构化、碎片化的特征。一方面，存算一体芯片本身具有异构性。芯片包含加速矩阵运算的张量核心以及加速向量计算的向量核心等加速部件。另一方面，虽然不同机构均基于 RISC-V 指令集进行芯片设计，但是不同机构的存算一体芯片具有迥异的架构特征，导致内部互联、存储器的存取方式等设计各不相同，造成了碎片化，给用户编程、程序优化带来了显著挑战，成为了当前人工智能芯片领域的国际性难题。

（二） 现有解决方案

为了解决上述问题，学术界和工业界提出了一系列的编译和编程解决方案。

- TVM[1] 是一个面向人工智能异构加速器的编译器框架，于 2018 年由华盛顿大学的研究团队发表。其以 TensorFlow、PyTorch 或 ONNX 等框架导入模型，将模型编译为可链接对象模块，然后轻量级 TVM Runtime 可以用 C 语言的 API 来动态加载模型，也可以为 Python 和 Rust 等其他语言提供入口点。在中间优化层级，其提出了 Relay IR 和 Tensor IR 两层中间表示来进行硬件无关（如常数折叠、算符融合等）、硬件相关（如计算模式识别与加速指令生成等）的优化。TVM 可以自动为多种硬件生成优化代码，支持端到端的学习优化，并且具备灵活的编译流程，但是 TVM 在面对新型加速器时，不但需要开发者根据芯片指令去扩展 TVM 中的 IR，还需要根据芯片的体系结构设计去添加定向优化策略，导致其扩展性较为有限。如 图 1 所示。

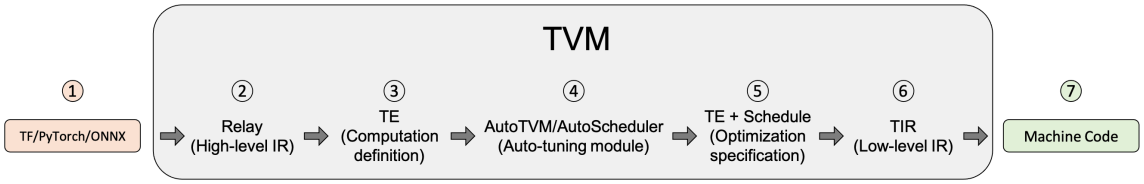

图 1　TVM 架构图

- Triton[2] 主要用于加速深度学习应用在 GPU 上执行效率的编译器，于 2021 年由 OpenAI 发布。Triton 是一种 Python DSL，用于编写机器学习内核，支持多种硬件，包括 CPU、GPU 和 ASIC 等等，其能够生成针对特定硬件优化的内核。在中间优化层级，Triton 编译器通过块级数据流分析技术，自动优化深度学习模型的执行过程。但是其主要面向英伟达和 AMD GPU 加速器，对 RISC-V 存算一体加速器支持有限。

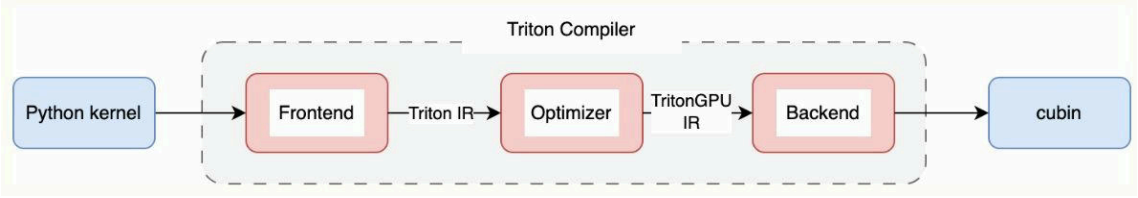

图 2　Triton 架构图

- XLA[3] 是谷歌于 2017 年开始开发的一种深度学习领域编译器。其接受来自 PyTorch、TensorFlow 和 JAX 等 ML 框架的模型，在中间优化层级，XLA 包括整体模型优化，如简化代数表达式、优化内存数据布局和改进调度等等。但是 XLA 主要针对 TensorFlow 优化，对其他框架的支持可能需要额外的工作；同时，其主要面向 GPU 和谷歌的 TPU，其中间表示为深度学习算子级别的抽象，使其难以拓展到 RISC-V 存算一体加速器。

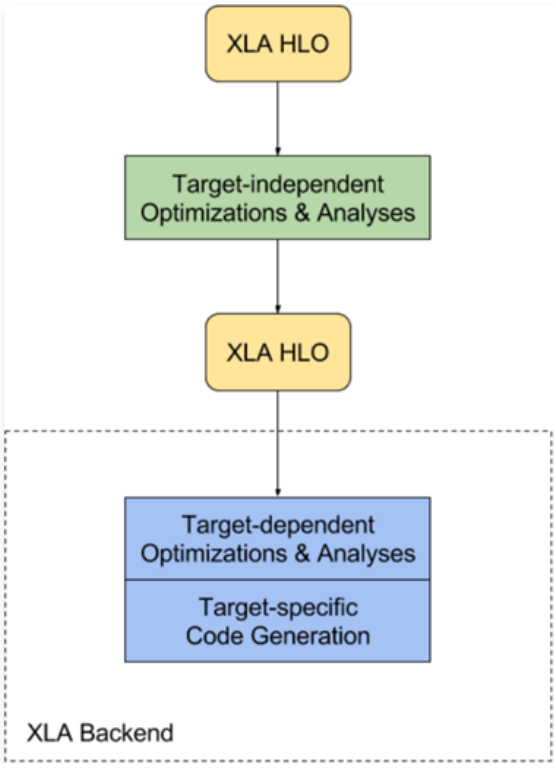

图 3　XLA 架构图

- IREE[4] 是一个开源的通用编译和运行时框架，由谷歌发布于 2019 年。通过输入高层次的机器学习模型，IREE 为各种硬件生成优化的可执行代码。在中间优化层级，IREE 利用 MLIR 进行多阶段优化，确保模型在目标平台上高效运行。IREE 提供了高性能的编译器后端，硬件抽象层允许轻松添加对新硬件的支持，但是 IREE 框架主要针对深度学习模型进行端到端的优化，而缺少统一编程模型。

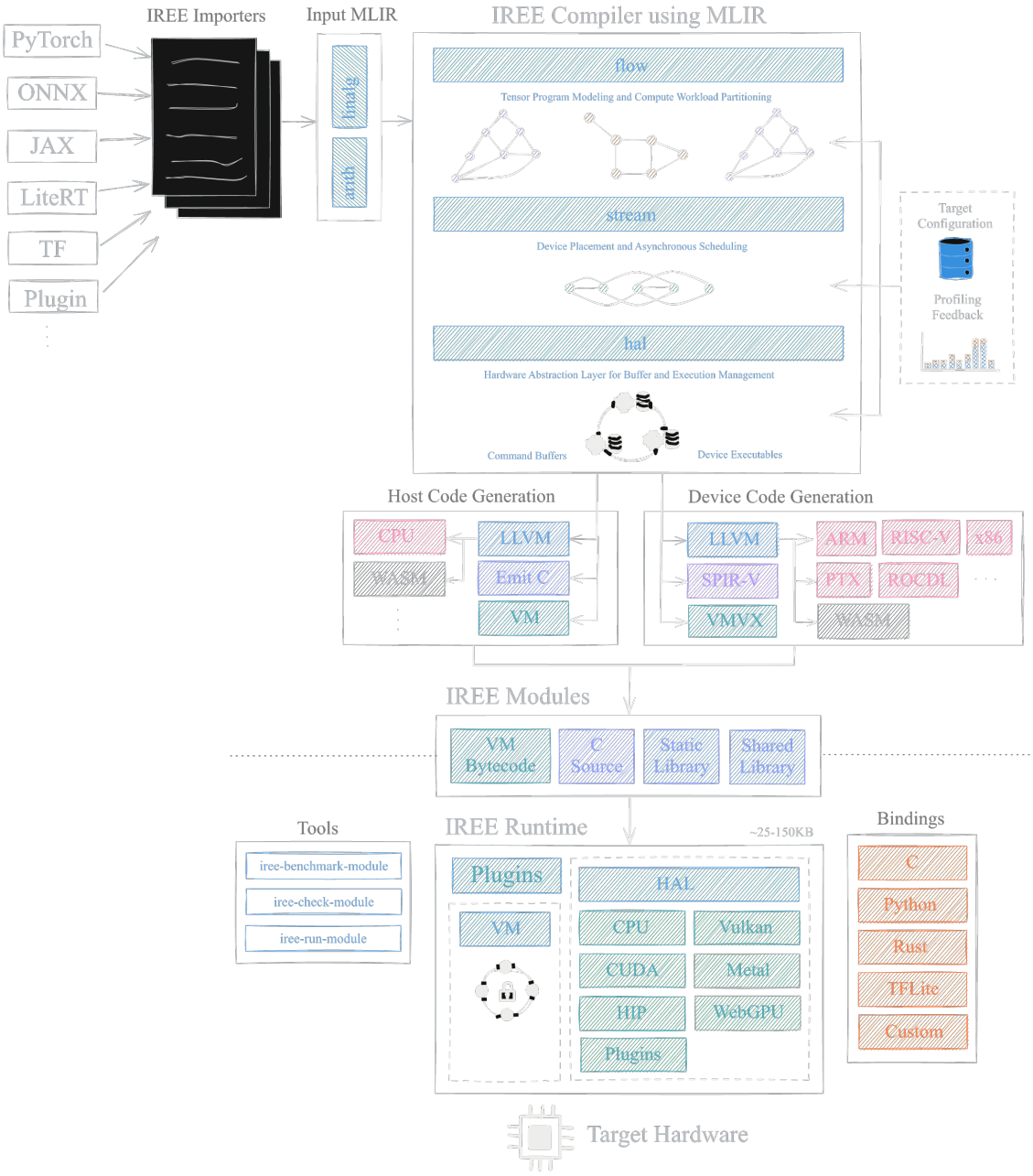

图 4　IREE 架构图

- AKG[5] 是一个由华为主导开发的深度学习编译器框架。AKG 可以接收来自 ML 框架的模型，生成针对特定硬件优化的内核。在中间优化层级，AKG 通过自动性能调优工具，自动生成优化的内核。AKG 提供了自动化的调优过程，可以显著提高性能，但是其主要支持华为的昇腾系列 AI 加速器和英伟达的 GPU，对 RISC-V 存算一体异构芯片支持并不友好。

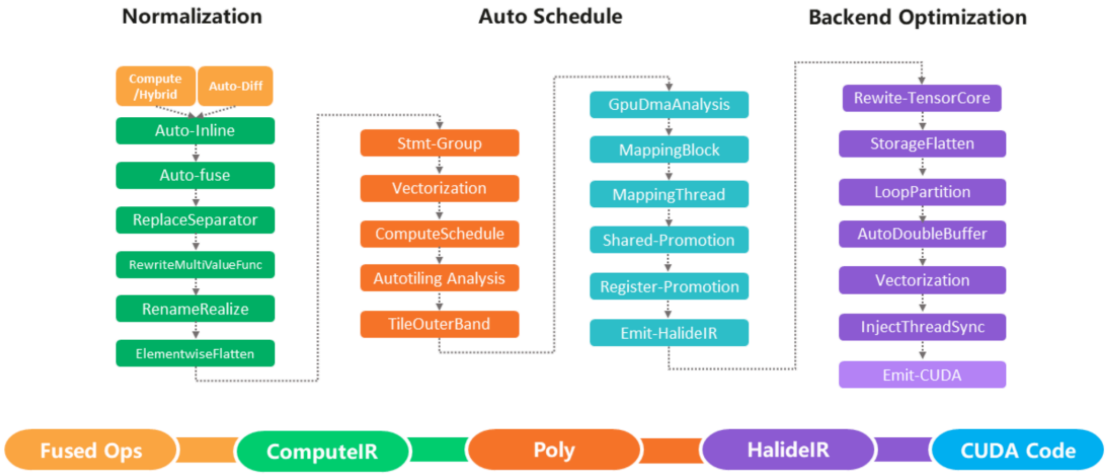

图 5　AKG 架构图

三、 理论与实践上的意义

（一） 理论意义

该项目的的理论意义在于探索如何通过解耦软件操作和硬件配置来提高人工智能应用的开发效率和部署便捷性。同时，通过深入研究 LLVM 编译器和 RISC-V 存算芯片的特性，探索新的动态编译优化方法，以适应人工智能应用动态变化和未来存算芯片架构特征的需求，更好地实现应用算子的自动映射和正确指令流生成，从而协调计算部件，挖掘芯片内部计算并行性。

（二） 实践意义

该项目的实际意义在于解决当前存算芯片异构化、碎片化现状下开发者面临的挑战，为 RISC-V 存算芯片提供有力的编译支持。通过修改 LLVM 编译器，实现对存算指令的支持，可以为存算一体芯片的开发和应用提供更高效的解决方案。该项目的实施将促进存算芯片的发展和应用，推动存算技术的创新和进步，提升 AI 应用的性能和效率，为实现存算一体架构的潜力提供重要支持。

四、 需要解决的关键理论问题和实际问题

（一） 理论问题

如何为解耦后的 RISC-V 生态提供编译支持，并允许编译器可以将应用算子自动映射到具有不同 IP 设计的加速部件，根据不同芯片架构特征生成正确的指令流来协调各个计算部件，挖掘芯片内部的计算并行性。

（二） 实际问题

如何将应用算子自动映射到具有不同 IP 设计的加速部件，同时根据不同芯片架构特征如何生成正确的指令流来协调各个计算部件，来挖掘芯片内部的计算并行性。

五、 基本方法、实验方案及技术路线的可行性验证

（一） 基本方法

本项目将面向 RISC-V 存算芯片修改编译器，实现对存算指令的支持。通过深入研究 LLVM 编译器的架构和工作原理，分析 RISC-V 存算一体芯片的特性，探索如何在 LLVM 中添加对 RISC-V 存算模拟器的支持。这包括但不限于对指令集的扩展、内存模型的适配以及优化策略的调整等，以实现应用算子的自动映射和正确指令流的生成，从而更好地协调计算部件，挖掘芯片内部的计算并行性，为 RISC-V 存算一体芯片提供有力的编译支持。

（二） 实验方案及技术路线

该项目面向 RISC-V 存算一体芯片修改 LLVM 编译器，并将其和底层 RISC-V 指令映射，以便在计算、计算部件协同、通信、存储互联、数据类型等多个角度对硬件特征进行抽象。 具体实验方案如下：

1. 项目准备
   - 文献调研：深入学习 LLVM 编译器架构、RISC-V 指令集及存算一体芯片（CIM）的相关文献，了解现有的研究成果和技术瓶颈。
   - 工具环境搭建：配置 LLVM 开发环境，获取并熟悉 RISC-V 的开发工具链，安装必要的模拟器和调试工具。
2. LLVM 编译器架构分析与扩展
   - LLVM 架构理解：详细分析 LLVM 的前端、中端和后端的工作流程，理解其插件机制和扩展接口。
   - 指令集扩展：根据 RISC-V CIM 芯片的存算操作需求，设计并实现新的指令集扩展。包括定义新的指令语义、编码格式以及相关的操作数。
   - 后端开发：在 LLVM 的 RISC-V 后端中集成新增的存算指令。实现指令的生成、调度和优化，包括 i 在汇编生成和机器码生成阶段的适配。
3. 内存模型适配
   - 存储器体系结构分析：研究 RISC-V CIM 芯片的内存架构，理解其数据存储与计算的交互方式。
   - 内存模型修改：调整 LLVM 的内存模型以适应 CIM 芯片的特点，确保数据在存储与计算单元之间高效传输。
4. 优化策略调整
   - 并行计算挖掘：分析 CIM 芯片的计算并行性，设计针对性的优化策略，如指令级并行、数据流优化等。

- 自动算子映射：开发自动映射机制，将高层次的算子（如矩阵乘法、卷积等）高效地映射到 CIM 芯片的存算指令，优化计算性能和能耗。
- 优化 Pass 开发：在 LLVM 的优化阶段引入新的 Pass，针对 CIM 架构进行特定的代码优化，如循环展开、向量化等。

5. 模拟器集成与测试
   - 存算模拟器继承：集成支持存算指令的 RISC-V 模拟器，确保新增指令能够在模拟环境中正确执行。
   - 测试用例设计：编写多样化的测试用例，覆盖新增指令、内存模型和优化策略，验证编译器的正确性和性能提升。
   - 性能评估：通过对比实验，评估修改后的 LLVM 编译器在 CIM 架构上的性能表现，包括执行速度、能耗和资源利用率等指标。
6. 文档撰写与总结
   - 实验记录与分析：详细记录实验过程、问题与解决方案，并对实验结果进行分析。
   - 报告撰写：整理项目成果，撰写本科毕业设计报告，涵盖研究背景、方法、实验结果与结论。

（三） 可行性分析

1. 技术方案可行
   - LLVM 的灵活性：LLVM 作为一个模块化、可扩展的编译器基础设施，具备良好的扩展能力，能够支持新增指令集和优化策略的集成。
   - RISC-V 的开放性：RISC-V 指令集架构公开透明，易于理解和扩展，适合作为研究与开发的平台。
2. 资源与工具支持
   - 丰富的开发资源：LLVM 和 RISC-V 均有完善的文档、社区支持和开源资源，有助于加快开发进度和解决技术难题。
   - 现有模拟器与工具链：本课题组已有基于 RISC-V 开发的 SRAM 存算一体 FPGA 模拟器，通过 RISC-V 扩展指令实现了矩阵乘操作。
3. 项目可分阶段进行
   - 模块化开发：项目可以分为指令集扩展、内存模型修改、优化策略调整等多个独立模块进行，降低开发复杂度，便于逐步实现和调试。
   - 渐进式集成与测试：通过逐步集成新增功能并进行测试，确保每个阶段的功能正确性和稳定性，降低整体项目风险。

六、应具备的条件及已具备的条件，可能遇到的困难与问题和解决措施

（一） 开展研究应具备的条件

课题组成员应熟练掌握编译器的相关实现、异构编程以及存算芯片的相关知识，需要对 MLIR 的设计方式较为熟悉。

（二） 已具备的条件

- 课题组成员在 2024 年全国大学生系统能力大赛编译系统实现赛中获得二等奖，熟悉编译器的基本理论和相关实现；
- 课题组成员具备一定的编程能力，能够使用 C++、Python 等语言进行编译器的开发。

（三） 可能遇到的困难与问题和解决措施

- 异构编程上手存在一定的难度，要加强理论学习，后续在老师的指导下，以项目为驱动，学习异构编程。

七、 论文研究的进展计划

1. 2024.11.01 ~ 2024.12.20：调研 AI 编译器、存算一体芯片等相关理论和研究现状，形成调研报告；
2. 2024.12.21 ~ 2025.01.19：配置 LLVM 开发环境，熟悉 RISC-V 开发工具链，并安装必要的模拟器和调试工具，撰写本科毕业设计开题报告；
3. 2025.01.20 ~ 2025.03.30：分析 LLVM 编译器架构同时进行扩展，并进行内存模型适配，准备中期检查相关材料；
4. 2025.04.01 ~ 2025.04.20：调整编译器优化策略并进行模拟器集成与测试，验证编译器的正确性和性能提升；
5. 2025.04.21 ~ 2025.06.10：梳理总结研究成果，撰写毕业设计报告，准备毕业论文答辩工作。

八、 课题所需器材、设备清单

- GPU
- RISC-V 存算一体模拟器

九、 参考文献

[1] CHEN T, MOREAU T, JIANG Z, 等. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning[C/OL]//13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)Carlsbad, CAUSENIX Association, 2018: 578-594. https://www.usenix.org/conference/osdi18/presentation/chen

[2] TILLET P, KUNG H T, COX D. Triton: an intermediate language and compiler for tiled neural network computations[C/OL]//Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming LanguagesPhoenix, AZ, USAAssociation for Computing Machinery, 2019: 10-19. https://doi.org/10.1145/3315508.3329973

[3] SNIDER D, LIANG R. Operator Fusion in XLA: Analysis and Evaluation[EB/OL] (2023). https://arxiv.org/abs/2301.13062

[4] iree[M]https://github.com/iree-org/iree

[5] ZHAO J, LI B, NIE W, 等. AKG: automatic kernel generation for neural processing units using polyhedral transformations[C]//PLDI 2021: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation2021