中国科学院大学
University of Chinese Academy of Sciences

# 硕士学位论文

## 面向 DIMM 近存计算系统的调度关键技术研究


作者姓名：潘炤伍

指导教师：张佩珩 正研级高级工程师

中国科学院计算技术研究所


学位类别：工学硕士

学科专业：计算机系统结构


培养单位：中国科学院计算技术研究所


2024 年 6 月

# Research on Scheduling Techniques for Near-Memory Computing Targeting real PIM system with PIM-DIMMs

**A thesis submitted to**

**University of Chinese Academy of Sciences**

**in partial fulfillment of the requirement**

**for the degree of**

**Master of Engineering**

**in Computer Architecture**

**By**

**Zhaowu Pan**

**Supervisor: Professor Peiheng Zhang**

**Institute of Computing Technology, Chinese Academy of Sciences**

**June, 2024**

## 中国科学院大学
## 学位论文原创性声明

本人郑重声明：所呈交的学位论文是本人在导师的指导下独立进行研究工作所取得的成果。承诺除文中已经注明引用的内容外，本论文不包含任何其他个人或集体享有著作权的研究成果，未在以往任何学位申请中全部或部分提交。对本论文所涉及的研究工作做出贡献的其他个人或集体，均已在文中以明确方式标明或致谢。本人完全意识到本声明的法律结果由本人承担。

作者签名：

日　　期：

## 中国科学院大学
## 学位论文授权使用声明

本人完全了解并同意遵守中国科学院大学有关收集、保存和使用学位论文的规定，即中国科学院大学有权按照学术研究公开原则和保护知识产权的原则，保留并向国家指定或中国科学院指定机构送交学位论文的电子版和印刷版文件，且电子版与印刷版内容应完全相同，允许该论文被检索、查阅和借阅，公布本学位论文的全部或部分内容，可以采用扫描、影印、缩印等复制手段以及其他法律许可的方式保存、汇编本学位论文。

涉密及延迟公开的学位论文在解密或延迟期后适用本声明。

作者签名：　　导师签名：

日　　期：　　日　　期：

# 摘 要

在过去的二十年里，随着通用处理器制程和工艺的演进, 面对计算密集型负载，近存计算技术是一种用于缓解，整体流程分，其核心是。xx 任务具有...，...，... 的问题，目前的，其... 有待提升。同时，由于，因此... 对... 有一定的要求。本文针对... 提出了基于... 的..，通过... 等技术在... 设备上进行高效的...，最终实现... 且保证... ... 的...。本文的主要研究工作以及贡献如下：本文提出了一种基于... 的...MetaPB。首先，得益于... 并行度高的优点，... 实现了在... 上的...。其次，在架构设计时，本文基于... 在保证... 的同时实现具备... 的...，并结合... ... 对模型进行改进。为了适应... 任务场景，本文设计并实验了高效的，在... 模块中使用了...，在... 阶段基于... 和... 技术进行... 预测。最终在保证模型效率的情况下，实现了一个具备较高.. 的算法模型。

本文在模型和架构层面实现了对... 算法的优化。对于模型层面，在整体框架设计时针对任务的特点引入结构，在模型内部使用... 提高了... 效率，最终提高了近存计算系统运行混合负载时的能耗与性能。为了适应不同... 应用场景下对运行能效和性能的需求，在可接受的... 损失条件下，本文使用... 技术实现了具备... 模型。在架构层面，本文探索了... 技术在... 任务场景中的应用，在不同... 下实现了在... 上的... 加速。最终本文联合和架构的... 优化，完成了对... 的整体改进。本文通过上述的算法设计并实现了一个基于... 的... ，该模型具备较高的...，同时不同... 的模型能够适应各类... 场景。本文提出的... 能够用于对... 的.. 进行...，通过...、...、... 和... 四个流程，得到具备较高... 的...。经过... 个测试... 的实验对比，本文提出的 MetaPB 相比于粗粒度 CPU-DPU 调度在性能优先模式上... 能效模式上... 平均获得了...

**关键词：**中国科学院大学，学位论文，模板

# Abstract

Chinese abstracts, English abstracts, table of contents, the main contents, references, appendix, acknowledgments, author's resume and academic papers published during the degree study and other relevant academic achievements must start with another right page (odd-numbered page).

**Key Words:** University of Chinese Academy of Sciences, Thesis, LaTeX Template

# 目 录

# 图目录

# 表目录

# 符号列表

## 字符

| Symbol | Description | Unit |
|---|---|---|
| $R$ | the gas constant | $m^2 \cdot s^{-2} \cdot K^{-1}$ |
| $C_v$ | specific heat capacity at constant volume | $m^2 \cdot s^{-2} \cdot K^{-1}$ |
| $C_p$ | specific heat capacity at constant pressure | $m^2 \cdot s^{-2} \cdot K^{-1}$ |
| $E$ | specific total energy | $m^2 \cdot s^{-2}$ |
| $e$ | specific internal energy | $m^2 \cdot s^{-2}$ |
| $h_T$ | specific total enthalpy | $m^2 \cdot s^{-2}$ |
| $h$ | specific enthalpy | $m^2 \cdot s^{-2}$ |
| $k$ | thermal conductivity | $kg \cdot m \cdot s^{-3} \cdot K^{-1}$ |
| $S_{ij}$ | deviatoric stress tensor | $kg \cdot m^{-1} \cdot s^{-2}$ |
| $\tau_{ij}$ | viscous stress tensor | $kg \cdot m^{-1} \cdot s^{-2}$ |
| $\delta_{ij}$ | Kronecker tensor | 1 |
| $I_{ij}$ | identity tensor | 1 |

## 算子

| Symbol | Description |
|---|---|
| $\Delta$ | difference |
| $\nabla$ | gradient operator |
| $\delta^{\pm}$ | upwind-biased interpolation scheme |

## 缩写

| | |
|---|---|
| CFD | Computational Fluid Dynamics |
| CFL | Courant-Friedrichs-Lewy |
| EOS | Equation of State |
| JWL | Jones-Wilkins-Lee |
| WENO | Weighted Essentially Non-oscillatory |
| ZND | Zel'dovich-von Neumann-Doering |

# 第 1 章　引言

# 第 2 章　相关理论与技术

# 第 3 章　研究方法与过程

## 3.1　研究方法

### 3.1.1　对 UPMEM 系统实际运行时性能瓶颈的归纳

### 3.1.2　具有可分性的跨平台负载类设计

针对近存计算核心在执行不同计算任务时性能差异大的问题，本课题综合以往异构计算的最佳实践，选用以算子为中心的负载表达方式，并重新设计了算子的封装形式，设计并实现了一种便于协同计算的跨平台算子基类，从而为算子提供跨平台、按比例调控不同部件所执行的任务量的能力，从而减轻了传统算子将任务完全错误分配到不合适的器件上以致产生负优化的可能。

由于使用算子所表达的计算任务高内聚低耦合，易于书写、维护和重用，且便于并行执行和调度，在工业界异构计算框架 [41], [42] 里正被广泛地采用，是一种被广泛验证的设计。单个算子作为一类具有相似计算特征的负载的共同抽象，可以被看作是一种输入向量到输出向量的映射。

然而，目前常用的算子在设计、优化和最终封装时往往针对的只是单一硬件后端，这就导致了：1. 算子作为负载的抽象，在设计时却并未考虑到多个异构计算器件的协同工作，转而只能使其依赖于上层逻辑中对异构设备中同一算子的分配和调用，这就在上层逻辑中增加了在算子调用、负载分配上的复杂度。2. 为了达到多器件协同，需要手动管理任务分配的比例以及器件间的同步，这与算子的设计初衷相违背。3. 涉及到多个复杂算子时，通过手动管理负载分配来充分发挥异构执行单元各自的优势，获得最优的性能十分困难。众多因素阻碍了在算子层面上异构计算器件的协同，从而导致了负载运行时异构器件的闲等和低利用率。

因此，本课题提出了一种可细分、同时可用于 PIM 和 CPU 的异构负载类设计。相似的设计如 PEI[23] 中，作者将近存计算算子与 CPU 指令一一对应并用硬件根据局部性决定指令派发，而在 SKMD[43] 中，作者使用 OpenCL 书写算子并分别编译 CPU 和 GPU 程序。考虑到 UPMEM 系统尚未存在对 OpenCL 的支持、且修改 CPU 硬件结构是不可行的，本课题借鉴两种设计中的长处，并针对本课题所基于的现实近存计算系统，设计了一种适用于 CPU 与 PIM 协同计算的、可按照指定比例进行任务划分的负载类，能够在不修改原有算子书写模式的同时，便于 CPU-PIM 协同计算时的任务分配与任务间调度，进一步提升运行效率。

### 3.1.3　调度策略生成模块设计

针对运行负载时总体计算资源利用率低的问题，本课题基于 3.2.1 中设计的混合负载类，设计了一种生成并优化 CPU-PIM 间任务分配比例的调度策略生成模块，以确定所有算子在不同器件上的任务分配比例，本质上是一种利用先验知

识进行任务分配的分治方法。

策略生成模块由量化器、回归拟合模块、策略价值判别模块及元启发策略优化器组成。其中：

1. 量化器用于在预热阶段收集算子在不同参数下的运行性能量化数据，并提供给回归拟合模块用于建立量化数据的一个回归模型。

2. 回归拟合模块用于接收量化器提供的数据，并根据其不同的参数进行回归拟合，并获得训练好的性能模型，本质上是一个通用的统计学习回归模型

3. 策略价值判别模块的核心是由一组涉及到传输、计算的时间及能耗的方程所构成的损失函数。该模块接收优化器所提议的调度策略，并传递给回归拟合模块来获取预测的量化信息，使用该信息以评估被提议策略的价值，最终将策略的价值评估提供给元启发策略优化器，以及（可选的）输出该策略在运行中预计情况下的量化信息，如时间和能耗。

4. 元启发策略优化器负责与策略价值判别函数交互，并在达到终止条件前为回归拟合模块提供新的策略提议，本质上是一个通用的元启发优化算法，参数空间的维度等于算子的数量，参数范围为 0 到 1 的闭区间（有一些算子如 load-store 只能由 CPU 执行）。

这四个模块中，量化器作为直接与负载进行交互的模块，会对每一个算子在不同参数下的性能数据进行尽可能详尽的测量，其对量化数据收集的完备性会影响使用该数据进行训练的回归拟合模块的拟合准确性，相对应地，在回归拟合模块中，对于合适回归算法的选用，会影响到最终训练出的回归模型的泛化性，而泛化性越好的模型在推理时得到的预期量化数据就会越贴近真实数据，使价值的判别更加具有参考意义。价值判别模块作为整个策略生成模块的核心，会根据预测的量化信息对该策略的价值做出评定，其中的价值函数的设计需要充分考虑对负载调优的偏好（性能或能耗），一个好的价值函数会使元启发优化算法更容易收敛并提供一个指标符合期望的调度策略。而元启发策略优化器作为对整个优化空间的一个搜索器，其收敛速度，搜索能力以及逃离局部最优解的能力都会影响整个调优过程的性能，并决定最后策略的优劣。

### 3.1.4 整体执行逻辑设计

(需大改) 针对负载间数据搬移代价高的问题，本课题采用了经典的计算掩盖传输的思路，但考虑到现实 PIM 系统中 MRAM 和 DRAM 非统一编址、且两者间传输带宽不高，为了进一步提高系统的计算效率并降低空泡，本课题利用 3.2.1 所设计的负载模块的可分性，将切分后的算子的每一个部分视作一个异步任务，该任务将在按照调度策略进行传输后，立即异步执行以及（按策略决定是否）写回，以细粒度划分任务的形式避免了传输时的同步停等，并提升传输总线在整个执行时间上的占空比，避免总线拥挤。

将按比例划分后的任务按照依赖关系，以生产者-消费者模型进行派发，每个器件处理不同比例的任务，通过该异步执行模块推进任务进展、避免频繁同步

和通信，以这种设计模式极大地减少了器件间闲等空泡，最大的利用了 CPU-PIM 的计算能力。

## 3.2　研究过程与具体实现

### 3.2.1　混合负载类建模与基于建模的工具链重构

#### 3.2.1.1　混合负载类建模

解决负载调度问题，首先要涉及到对负载本身的建模，也要同时考虑到运行负载的系统：

在 UPMEM PIM 系统运行近存计算负载的关键步骤是 PIM 程序的交叉编译和 PIM-MRAM 到 DRAM 间数据流的控制。以简单的向量加（Vector-Add，VA）算法为例，典型的执行流程是：1. 交叉编译运行在 PIM 核心上的代码 2. 使用 UPMEM API 分配若干个可用的 PIM DPU 3. 在 CPU 部分的代码中为 PIM kernel 开辟输入输出缓冲区 4. 将交叉编译后的代码载入 PIM 核心 5. 传输参数和缓冲区数据到指定的核心 6. 启动 PIM 端代码的执行 7. 同步等待执行完毕并传回数据至主存储区的输出缓冲区。

在仅执行单一 kernel 或若干个无数据依赖的 kernel 时，该流程可以很好的完成任务，但多阶段任务中，由于现实中 PIM-MRAM 和 DRAM 并不统一编址，对 PIM 在不同 kernel 间 MRAM 上下文的保存和使用就成为了减少 MRAM-DRAM 通信的关键，然而在被广泛使用的流程中，kernel 本身由于提前被静态编译为二进制文件并载入，是无法利用运行时的调度信息（原地使用上下文继续计算或清理堆内存、等待 CPU 传输后计算）和可变参数（缓冲区大小等）的，这不利于负载在有上下文情况下的驻留或调度。

故在实现适用于负载调度系统的抽象负载类时，本课题的设计如下：

考虑到 PIM 元件间通信需要经过主存，无直接互联、通信开销大，本设计中所有的算子的书写都必须满足可分的前提，可以拆分为若干个任务块，且对应特定的、无重叠的输入输出数据块。

考虑到 PIM 元件访存延迟比 CPU 低，但从 CPU 端访问 PIM 核心需要通过内存通信（非统一编址，访问开销大），所有负载对象在设计时会同时包含 PIM kernel 和 CPU kernel，分别用于执行驻留 PIM 上的代码，以及在 CPU 上执行相同功能的代码。

考虑到使用 PIM 核心时需要预先编译其二进制，并在调用 kernel 前传入 PIM 元件，难以动态定制任务执行的逻辑。本设计中所有 PIM kernel 在调度策略决定后在宿主机调度协程中进行传参并编译，所有同一任务下切分的负载对象共享相同的二进制。

至此，如图一个可以将任务划分为若干个负载块的负载抽象就建立完毕了，该负载类同时包含可以在 CPU 和 PIM 上执行的 kernel，以及控制其具体执行逻辑的上层控制函数（control wrapper function），其中上层控制函数和调度器类是

友元关系，可以根据调度器的决策来进行数据传输后执行、原地执行或上下文执行，从而在指定的硬件执行对应的算法。

#### 3.2.1.2 基于负载建模的工具链重构

由于 UPMEM-PIM DPU（以下简称为 DPU）和通用处理器的体系结构不同，实际运行在 DPU 上的二进制程序将使用被特殊定制的 C 标准库和自定义原语，故所有 DPU 的程序都需要在 UPMEM 提供的、基于 CLANG-LLVM 框架开发的特殊 C 编译器中进行编译。而宿主端的程序可由普通的 C、C++、Python 或 JAVA 书写并使用通用编译器/解释器进行编译或解释运行。通过直接或间接地调用 C API 中的函数，宿主机端程序可以在运行时用指定编译好的 DPU 二进制文件绝对路径的方式，将由特定编译器编译好的 DPU 二进制代码动态加载到指定的 DPU 上，并控制数据的传输以及 DPU 程序的执行，而有关于算子数据类型、DMA 参数等与宿主机公用的信息也都在编译时确定并再也无法进行变动，更无法在运行时进行调优。

在编译构建整个项目的时候，一种常用的做法就是书写一个特定的 Makefile 并手动管理依赖关系，将 DPU 和宿主机代码通过规则中调用 shell 命令的方式控制编译参数与二进制文件存放位置。该方法本质上是书写一个复杂一些的脚本。这种分离编译并传入二进制绝对路径执行的做法，自 Prim-benchmarks 之后，在众多基于该系统的项目中都在沿用，虽说在操作上和理解上难度不大，但是在应对多算子、多数据类型甚至是运行时泛型算子的情况时会十分繁琐，而这些情况在调度框架中是十分常见的，故本课题在该方面做了以下的工作：

(a). 用 C/C++ 同时兼容的语法书写 Consensus.h 头文件，在其中包含对数据类型标识、公用结构体等的宏定义，最大限度减少编译期固定的参数定义，而将可以在运行时确定的参数以结构体的形式传入 DPU，保留灵活性

(b). 受 tag_invoke[44] 的启发，在 Consensus.h 中，使用宏编程在 C 语言中仿照实现了高级语言里的类型反射行为，通过在宿主机和 DPU 间通过共有的类型标识传递类型信息，使得 DPU 可以在硬件支持的十余种数据类型、以及 Consensus.h 中声明的复合类型范围内支持泛型算子的执行，其结构大致如图 1 所示。

(c). 采用现代化的 CMake 构建系统构建整个项目，为项目的混合编译编写了特殊的 CMakeLists，自动管理宿主机和 DPU 程序的编译方式以及二进制文件路径，并暂时使用 C++17 标准的 filesystem 库进行运行时路径解析，避免使用绝对路径索引 DPU 二进制文件，提升鲁棒性，该项目目前的目录树如图 2 所示。当前正在进行的工作中还包括宿主机代码编译期 DPU 二进制文件嵌入行为的编写，在该项工作完成之后，Release 版本的近存计算程序中宿主机和 DPU 端代码由路径索引的二进制依赖关系将不复存在，将更有利于程序的移动和部署。

考虑到整体性能和开发效率，本课题选用现代 C++ 语言作为宿主机编程语言，这就意味着需要和宿主机 C++ 开发库进行交互。

UPMEM 官方所提供的通信、执行库的编写语言和原生接口都是以 C 的形式提供的，官方提供的其余语言 API，如 C++、Python、JAVA 接口都是由 C 语言接口进一步封装而得来，本课题需要使用的 C++ API 在实际使用中暴露了多个问题：

(a). 健壮性问题（数据类型越界、内存泄漏）

(b). 继承关系混乱（不规范的派生关系）

(c). 成员函数重载过多且封装不良，不利于分析

综上，本课题弃用原始的 UPMEM C++ API 并重新封装原生的 C 接口，目前已将原来作为类成员的通信方法分离并重构完毕，构造成单独的、可量化的算子并继承自统一的算子基类（如图 3），同时在基类中引入全局的 ChronoTrigger 指针（见工作 3），使一切有关于计算和传输的行为都可以被统一调度和量化。

### 3.2.2　调度策略生成与优化模块实现

#### 3.2.2.1　运行时量化模块 ChronoTrigger 的设计与实现

该量化器的底层通过 Intel PCM 直接在运行时读取系统硬件性能计数器，从而获取暂态或一段时间内的性能信息，本实现将其抽象为一个单例模式的“探针管理器”，通过 tick-tock（如 4-6）的方式对程序中某一部分进行量化数据的采集，并使用哈希表的方式将测试时手动传入的程序段名称映射到指定的 Report 数据结构中，该结构里包含着若干种性能量化指标的统计量。

在复杂度和精确性上，ChronoTrigger 对性能指标统计量的计算（如均值、方差）都是增量式的，不需要遍历并存储历史性能数据，在时间和空间上的开销均为常数量级，便于后期动态调度时对统计量的即时分析，且本量化器会在构造时进行 100 次的 tick-tock 空转并生成一个 __BIAS__ 报告项，用于在输出时纠正其他报告中存在的系统偏差。

#### 3.2.2.2　性能量化数据回归模块的实现

#### 3.2.2.3　策略评估及元启发策略优化模块的实现

由于本课题涉及到高维空间内复杂函数的最优解寻找，且对其的直接求解是 NP 完全问题，故采用元启发优化算法对全局最优解进行逼近。

截至目前，本工作已复现了 3 种元启发算法，分别为同、异步的粒子群算法 [45]（Particle Swarm Optimization，PSO），算术优化算法 [46]（Arithmetic Optimization Algorithm，AOA）和爬行优化算法 [47]（Reptile Search Algorihm，RSA），并以 C++ 泛型类的形式将其书写完毕，在实现中使用泛型函数指针，将目标函数的定义与优化器分离，可以对自定义决策质量判别函数进行最优解的求取。在验证后，于若干个高维、多局部最优点的函数优化中取得了与原论文中接近的效果，并获得了较好的性能。现已将该三种算法封装完毕并设计继承了统一的优化器接口以供主项目调用

### 3.2.3 策略执行逻辑实现

由于课题将近存计算系统看作是一种特殊的异构计算设备，且该设备和 CPU 间的阻塞传输对整体性能的影响较为严重，所以在合理划分负载分配比例的工作上，要想进一步提升整个调度系统的性能，一个异步的传输-执行模块对于提升该系统的性能是十分重要的。经过试错与总结，本课题最终选用未来将引入至 C++26 标准库的 stdexec 提议（P2300），以结构化异步并发（Structured Asynchronous Concurrency）的设计思路来构建整个异步传输-执行模块。

在结构化并发的设计思路里，所有的异步计算主要由四种概念所支撑，其关系可见图 4-8：

Sender（或译为发送者）：是对于异步计算工作的描述，可以被认为是计算图中的一个计算节点，会包含一个用于连接 Receiver 的方法，用于和自己工作对应的接收者建立联系。

Receiver（或译为接收者）：是对传统异步计算中“回调（Callback）”的一种抽象，具有三种接受发送者信息的方法，分别用于接受异步工作的正常返回值、错误信息或提前终止信号中的一种。

Operation_state（或译为工作状态）：是真正存储工作上下文的一个类，具有启动异步任务的能力，与每一对 Sender-Receiver 都是一一对应的。

Scheduler(或译为调度器)：调度器是一个轻量级句柄，代表一种将工作调度到执行资源的策略。调度器概念由一个单一的、返回 Sender 的方法定义，这个返回的 Sender 将使依赖于该 Sender 的后续有任务调度在由调度器确定的执行资源上完成。

本课题在抽象负载类的设计（见 3.2.1）中，考虑到传输同步的开销，要求所有任务都是可以切分成若干个子任务并进行拼接完成计算，而在 P2300 中，所有异步执行的任务都被建模为一个 Sender（发送者），而 Sender 可由若干个 Sender 所构成，在结构上是自相似的，在这一点上与本设计匹配，可以用较少的额外工作进行负载类的重构。

在实现中，所有 Sender 所表示的工作都被定义是懒惰 (lazy) 的，在没有显式执行同步获取以最终结果前，所有的异步执行任务的声明，都只是在建立依赖关系以及构建回调链条。再加上 Sender 的底层实现是通过模板嵌套的形式，大多数 Sender 算法本身就具有可分和可组合性，编译器能够看到使用 Sender 描述的一系列工作形成了一棵尾调用树，从而实现了内联化并消除了大部分内部机制所带来的开销。所以它们具有在编译时期利用编译器将多个算子融合在一起并形成 Sender 链的机会，进而结合可分的算子构造，完成多个算子的自动融合。

本课题完成了基于 P2300 的 Sender-Receiver 进行了线程池性能测试实验，在未来的补充实验里，会使用 Sender-Receiver 模式对整个执行流程进行异步化加速。

至此对本课题所设计的调度逻辑的叙述就已经完成了，关于运行本调度算法的一个典型示例以及伪代码如图 4 9：

## 3.3　本章小结

# 第 4 章　实验验证与结果分析

## 4.1　实验验证

实验部分，讲述 benchmark 的选取以及测试标准

### 4.1.1　数据集

#### 4.1.1.1　算子集合

#### 4.1.1.2　计算图集合

### 4.1.2　实验评价指标

## 4.2　数据分析

## 4.3　本章小结

# 第 5 章　总结与展望

5.1　本文工作总结

5.2　下一步研究方向

# 参考文献

# 附录　公式

# 致　谢

此处填写致谢。

2023 年 6 月

# 作者简历及攻读学位期间发表的学术论文与其他相关学术成果

## 作者简历：

××××年××月——××××年××月，在××大学××院（系）获得学士学位。

××××年××月——××××年××月，在××大学××院（系）获得硕士学位。

××××年××月——××××年××月，在中国科学院××研究所（或中国科学院大学××院系）攻读博士/硕士学位。

工作经历：

## 已发表（或正式接受）的学术论文：

(1) 已发表的工作 1
(2) 已发表的工作 2

## 申请或已获得的专利：

（无专利时此项不必列出）

## 参加的研究项目及获奖情况：