面向 MT-3000 的软件管理 SM 缓存优化框架

李怀锦[1)] 董小社[1)] 曹骏恺[2)] 窦水韬[2)] 刘宇航[1)] 王 强[3)] 白秀秀[2)]

[1)](西安交通大学计算机科学与技术学院 西安 710049)

[2)](西安交通大学软件学院 西安 710049)

[3)](西安交通大学网络信息中心 西安 710049)

摘 要 针对 MT-3000 众核处理器访存中存在的访存特征表征粗粒度、优化策略通用性不足、缺乏自动化优化框架问题，提出一种特征驱动的 SM(Scalar Memory，标量内存)缓存加速访存优化框架。该框架包含三个核心模块：基于 Clang AST 的访存特征提取模块 MACE-C，通过动静结合分析实现变量级访存行为的精确刻画；自适应缓存管理策略库 MT-cache，提供批量缓存、单缓冲区和直接映射三种互补策略；基于局部性感知与密度加权的策略推演模型 LADWIM，实现从访存特征到缓存配置的自动化推导。在 PolyBench 基准测试集上，框架对线性代数类算法最高可达 73 倍加速比；在航空发动机内流数值模拟应用 X-Blue 上，实现了单节点 87%的整体性能提升，在万核规模下性能提升 97%，与同类传统手工优化方法相比，本文框架的加速比提升最高约 72.9%。实验结果表明，该框架能够有效降低访存优化门槛，为我国众核处理器的高性能计算应用提供了系统化的优化方案。

关键词 MT-3000；众核处理器；访存优化；特征驱动；缓存管理

中图法分类号 TP302 **DOI 号** 10.11897/SP.J.1016.2026.01478

Software-Managed SM Cache Optimization Framework for MT-3000

LI Huai-Jin[1)] DONG Xiao-She[1)] CAO Jun-Kai[2)] DOU Shui-Tao[2)]
LIU Yu-Hang[1)] WANG Qiang[3)] Bai Xiu-Xiu[2)]

[1)](School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049)

[2)](School of Software Engineering, Xi'an Jiaotong University, Xi'an 710049)

[3)](The Network Information Center, Xi'an Jiaotong University, Xi'an 710049)

Abstract With the continuous growth of high-performance computing demands, many-core processors have become a crucial technological direction for solving large-scale parallel computing problems. However, the widening performance gap between processor computational capabilities and memory access speeds has created the "memory wall" problem, which severely constrains overall system performance. As a representative of domestic heterogeneous many-core processors, MT-3000 adopts a software-managed on-chip memory architecture, and its complex memory hierarchy presents significant challenges for memory access optimization. To address three critical challenges in memory access optimization research for MT-3000 many-core processors—coarse-grained memory access characterization, insufficient generality of optimization strategies, and lack of automated optimization frameworks—this paper proposes a feature-driven SM (Scalar Memory) cache acceleration framework for memory access optimization.

收稿日期：2025-08-13；在线发布日期：2026-03-11。本课题得到国家重点研发计划(No. 2023YFB3001804)资助。**李怀锦**，硕士，主要研究领域为高性能计算、计算机体系结构。E-mail：lihuaijin@stu.xjtu.edu.cn。**董小社**，博士，教授，博士生导师，主要研究领域为高性能计算、高效能存储系统、云计算。**曹骏恺**，硕士研究生，主要研究领域为高性能计算。**窦水韬**，硕士研究生，主要研究领域为高性能计算。**刘宇航**，硕士研究生，主要研究领域为高性能计算。**王 强**，博士研究生，主要研究领域为高性能计算、计算机体系结构。**白秀秀(通信作者)**，博士，副教授，主要研究领域为高性能计算、计算机视觉、人工智能。E-mail：xiubai@xjtu.edu.cn。

The proposed framework comprises three core modules. First, MACE-C, a memory access characteristic extraction module based on Clang AST, combines static code analysis, dynamic runtime monitoring, and statistical feature analysis. Through source-code-level instrumentation techniques, it captures the memory access address sequence of each variable, computes key metrics including access stride distribution, access density, and spatial locality, thereby achieving precise characterization of variable-level memory access behaviors throughout their entire lifecycle. Second, MT-cache, an adaptive cache management strategy library, designs three complementary software cache management strategies tailored to MT-3000 architecture characteristics: bulk caching for high-density small data blocks, single-buffer for sequential access patterns with good locality, and direct mapping for irregular access patterns. Through unified macro interfaces that encapsulate underlying DMA communication and cache replacement logic, MT-cache effectively reduces programming complexity for developers. Third, LADWIM (Locality-Aware and Density-Weighted Allocation Strategy and Parameter Inference Model), a strategy inference model based on locality awareness and density-weighted allocation, constructs memory access feature vectors, calculates spatial partition factors, and executes multi-variable joint optimization algorithms to enable fully automated derivation from memory access characteristics to cache strategy selection and parameter configuration, eliminating dependence on manual expertise.

Comprehensive experimental evaluations were conducted on both benchmark suites and real-world applications. Experimental results on the PolyBench benchmark suite demonstrate that the framework achieves up to 73× speedup for linear algebra algorithms, with significant performance improvements also observed for data mining and convolution algorithms. On the X-Blue aero-engine internal flow numerical simulation application, the framework realizes 87% overall performance improvement on a single node and achieves 97% performance enhancement at the ten-thousand-core scale in distributed computing environments, validating the framework's excellent scalability. Compared with similar traditional manual optimization methods, the proposed framework achieves a maximum speedup improvement of approximately 72.9% while significantly lowering the optimization barrier. The strategy selection validation experiments confirm that LADWIM correctly identifies the optimal caching strategy for variables with different access patterns, and parameter inference validation shows that the model-recommended configurations achieve performance within 0.5% of exhaustive search results. Furthermore, the framework demonstrates good architectural adaptability and can be extended to other many-core processor platforms with software-managed on-chip memory, such as the Sunway SW26010 series processors.

The experimental results indicate that this framework effectively addresses the high threshold and low generality problems of memory access optimization on the MT-3000 platform, providing a systematic optimization solution for high-performance computing applications on domestic many-core processors.

Keywords　MT-3000; many-core processor; memory access optimization; feature-driven; cache management

1　引　言

随着高性能计算需求的不断增长,众核处理器架构已成为解决大规模并行计算问题的重要技术方向。然而,处理器计算能力与访存速度之间的性能差距日益加剧,形成了制约系统整体性能的“内存墙”问题[1]。在众核环境下,多个计算核心同时竞争有限的内存带宽,使得访存优化成为影响系统整体性能的关键因素。

面对这一挑战，学术界和工业界开始探索各种解决方案。从技术路径来看，主要分为硬件层面的架构创新和软件层面的优化技术两个方向。在硬件层面，众核处理器通过增加核心数量和引入复杂的内存层次结构来提升计算能力；在软件层面，则通过访存模式分析、缓存管理优化和数据局部性改进等技术来缓解内存墙问题。其中，众核处理器因其强大的并行计算能力和相对较低的功耗，已成为高性能计算领域的主流选择。

在我国众核处理器的发展中，MT-3000 作为异构众核处理器的代表之一，采用了片上异构众核架构设计，集成了通用计算核心和加速计算核心，并配备了层次化的内存系统[2]。这种复杂的内存层次结构为高性能计算提供了基础，但同时也带来了访存优化方面的挑战。

通过对现有研究的深入分析，我们发现当前针对 MT-3000 平台的访存优化研究主要集中在特定应用的手动优化上[3-5]，这种研究现状暴露出三个关键问题：

(1)访存特征表征粗粒度：现有方法主要关注程序整体的访存特征，缺乏对单个变量或数据结构级别访存特征的精确描述[6-7]。这种粗粒度的特征表征限制了对程序内部不同数据对象访问模式的理解，现有的细粒度分析多依赖人工经验，需要专业知识支持，缺乏自动化和普适性，难以为针对性优化提供细粒度的数据支持。

(2)优化策略通用性不足：大多数研究采用针对特定应用场景的定制化优化方案，缺乏通用性和可移植性。这种“一应用一方案”的优化模式不仅增加了应用开发的复杂度和周期，也限制了优化经验的积累和推广，使得每次面对新应用时都需要重新进行深入的架构分析和优化设计。

(3)缺乏自动化优化框架：从访存特征到缓存配置的映射过程高度依赖人工经验，缺乏理论指导的自动化推演方法。当前的优化过程通常需要开发者具备深厚的计算机体系结构知识，能够理解复杂的访存模式并设计相应的缓存管理策略，这一高门槛限制了优化技术的普及和应用。

为解决上述问题，本文提出了一个面向 MT-3000 众核架构的特征驱动缓存加速访存优化框架。该框架通过自动化的访存特征提取、缓存管理及策略推演，为 MT-3000 平台提供了系统化的访存优化方法。

本文主要贡献包括：

(1)设计并实现了基于 Clang AST 的访存特征提取方法 MACE-C(Memory Access Characteristic Extraction Module Based on Clang AST)，采用动静结合的分析方法，实现了对程序中单个变量全生命周期访存行为的精确刻画；

(2)构建了针对 MT-3000 架构特性的自适应缓存管理策略库 MT-cache，提供批量缓存、单缓冲区和直接映射三种互补的优化策略，有效封装了底层并行编程模型的复杂实现细节；

(3)提出了基于局部性感知与密度加权分配的策略推演模型 LADWIM(Locality-Aware and Density-Weighted Allocation Strategy and Parameter Inference Model)，通过理论驱动的方法实现了从访存特征到缓存配置的自动化转换；

(4)通过实验验证了框架的有效性，在基准测试和实际应用中均取得了显著的性能提升。

论文其余部分组织如下：第 2 节回顾相关工作；第 3 节介绍框架设计与实现；第 4 节展示实验评估结果；第 5 节总结全文并展望未来工作。

2 相关工作

2.1 MT-3000 异构多核处理器架构

MT-3000 处理器由国防科技大学设计实现，在 1.2 GHz 工作频率下，可达到 11.6 TFLOPS 的双精度浮点性能和 45.4 Gflops/w 的能效比[2]。

如图 1 所示，MT-3000 处理器具有四个主要设计特点：首先是异构多域微架构，将 16 个 CPU 核组织为通用域，96 个控制核和 1536 个加速核组织为加速域，并将加速域均分为四个自治簇；其次是结合 VLIW 与加速阵列的微架构，每个加速核以 VLIW 方式工作，每 16 个加速核与一个控制核组织成一个加速阵列；第三是超高带宽片上向量存储，向量存储支持最多两个向量 load/store 操作，可为 16 个加速核提供 16×2 个双字数据；第四是层次化互连网络，采用 mesh 网络连接 16 个 CPU 核并支持缓存一致性，采用 crossbar 连接超级节点，八条高带宽数据通路连接通用域和加速域[8]。

在内存系统设计方面，MT-3000 采用了系统化的混合内存层次结构。通用域主要负责运行操作系统和管理四个加速簇，每个 CPU 核都拥有私有的两级缓存系统，L2 缓存容量为 512 KB，采用 16 路组相连，缓存行大小为 64 字节。加速域被均分为四个簇，各个簇可独立运行，簇间通过通用域进行通信。

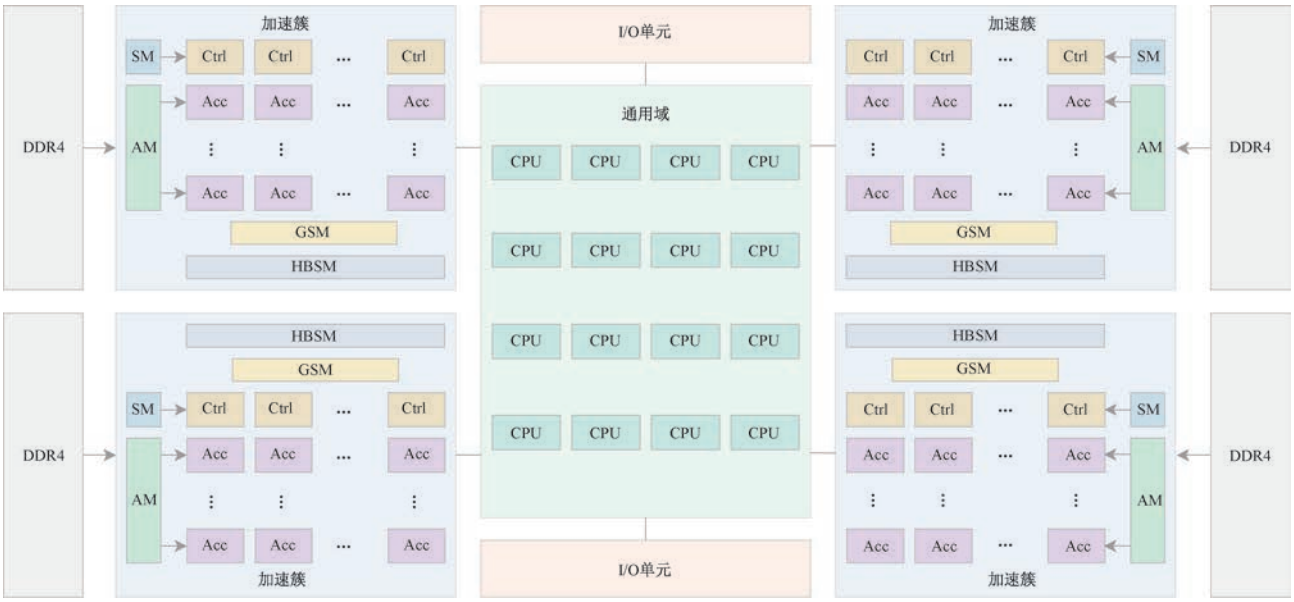

图 1　MT-3000 众核架构示意图

加速簇承担 MT-3000 上的大部分计算任务,簇内有 24 个加速阵列,每个加速阵列包含一个控制核和 16 个加速核,以 VLIW 方式进行组织。

如图 2 所示,加速簇采用混合式内存架构设计,每个加速簇在片外有 32 GB DDR4 主存,提供 51 GBps 带宽。在片上共有四层存储,48 MB 的 HBSM 采用多 bank 设计,被通用域和簇内所有核心共享,带宽达 307 GBps;6 MB 的 GSM 被当做私有的片上内存,只被簇内核心共享。簇中的每个核内有寄存器、SM(Scalar Memory,标量内存)和 AM(Array Memory,向量内存)三种存储,其中 SM 大小为 64 KB,控制核私有,用于标量数据存储;AM 大小为 768 KB,加速核私有,用于向量数据存储,支持半字、全字和双字三种数据类型,实现基于 CRC 的错误检测与纠正算法,并支持非对齐向量访问。

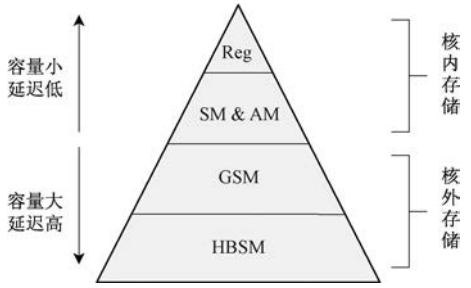

图 2　MT-3000 加速簇片上存储层次

软件方面,MT-3000 采用四层软件架构,如图 3 所示。基础层为 Linux 操作系统,通过自定义设备驱动管理通用域与加速域之间的通信,使用 MT-3000-gcc 编译器将 C 代码编译为可执行二进制文件。libMT 提供加速域低级管理接口,HPML 提供针对 MT-3000 优化的高性能数学库。在此基础上,MT-3000 提供了异构编程模型的 hthreads 和兼容 OpenCL 1.2 标准的 MOCL3,为开发者创建了完整的编程环境[8-9]。

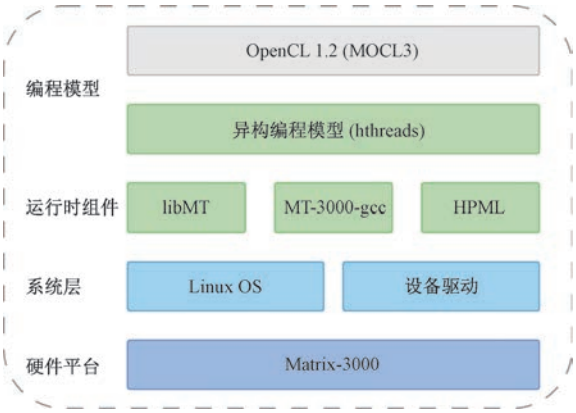

图 3　MT-3000 软件架构示意图

这种异构多域处理器设计理念有效管理了大规模异构核心,降低了数据移动成本和控制开销,为高性能计算提供了新的解决方案。然而,其复杂的内存层次结构和软件管理的片上存储也为访存优化带来了挑战,需要开发人员对底层架构有深入理解,导致编程门槛较高。

2.2　面向 MT-3000 的 SM 访存优化

近年来,针对 MT-3000 处理器的访存优化研究在高性能计算、人工智能、大数据等领域展开,主要从数据布局、缓存管理、预取机制以及网络通信等方面入手。

张元胤等[10]在 MT-3000 平台上实现等值线与等值面提取算法，进行了基于多游标的访存优化，以较少次数的比较操作快速定位各个等值点在数组中的位置，同时采用了等分缓存管理策略，将 SM 空间划分为多个等份，分段循环加载内存中的属性数据到 SM 上，以插值方式计算各个等值点坐标，可实现超过 4 倍的加速，并且性能随核心数的增加呈近线性扩展，通过分段循环加载和缓存填充来避免大量随机访问，是针对特定可视化算法的手工优化典型案例。

Chen 等[3]针对格点 QCD 算法，为容量有限的 AM 设计了合理的数据排布方式，高效利用 AM 进行向量化计算，同时利用 GSM 优化 z 维和 x 维的边缘数据核间通信，实现了大规模并行下显著的访存性能提升。Dslash 算法和最小残差法在 DSP 的加速阵列上均实现了良好的加速效果，与原始的单核串行程序相比，前者实现了 31 倍的性能提升，后者实现了 14.4 倍的性能提升。

路瑶等人[11]针对 MT-3000 异构处理器的 MHA 计算优化，通过算子融合将 Linear、Softmax 等算子合并执行，减少中间数据存储。在访存优化方面，利用广播机制将权重矩阵一次性加载到片上，通过 GSM 缓存中间注意力得分，消除冗余 DDR 传输。优化后 Linear 算子单簇性能达 1.53 TFLOPS（占理论峰值 37.7%），在典型大模型配置下（嵌入维度 4096/8192，头数 32/64）相较 V100 GPU 实现最高 23.53 倍加速。

王昊天等人[12]实现了 MTTorch 扩展库，针对 Transformer 核心算子进行优化。对于规约算子设计了多核 Welford 迭代算法，利用 GSM 进行核间数据规约。访存优化采用乒乓算法，在 AM 上申请双倍缓冲空间，在执行当前批次计算时通过异步 DMA 预加载下一批数据，实现计算与传输重叠。实验表明，数据规模超过 2^{20} 时乒乓算法带来显著性能提升，单 DSP 簇上核心算子相较 CPU 实现 2～8 倍加速。

Cheng 等[13]在 MT-3000 芯片上进行 FFT 计算优化时，通过配置 DMA 的 64 位传输控制字、事件使能寄存器和 SG 数据传输控制寄存器，实现从不规则内存位置到规则目标位置的高效数据收集，通过中断机制实现了连续的双通道传输。此外，通过 DMA 双通道传输，在处理一批数据的同时预加载下一批数据，形成双缓冲机制，实现了计算与数据传输的并行。优化后所有相关 FFT 函数的性能均提升了 99%以上。

Li 等[4]在分子动力学模拟移植和优化研究中，在 SM 上利用缓存替换策略对 Rho 函数进行加速，当访问插值点时检查缓存是否缺失，进而计算新的偏移量进行下一批数据的输入。同时，通过 DSP 加速核的矢量化和空间排序算法，大幅减少了随机访存次数与 DMA 传输次数，显著提高了 MT-3000 上的访存性能，使单节点性能提升近 6 倍，实现了优秀的强扩展性和弱扩展性。

Li 和 Liu 等[14-15]在 SN 离散坐标算法优化研究中，通过将球谐展开系数等标量数据和体积积分源矩等向量数据分别利用异步 DMA 通信加载到 SM 和 AM 中，形成软件缓存技术，显著降低了访存延迟。此外，通过提出两级 KBA 策略和异构并行通信优化，提升整体计算效率，实现了良好的扩展性与计算效率，在单加速阵列上的性能效率接近理论上限，在 S14 正交集和 32×32×64 空间尺度下，双精度性能达到每加速阵列 25.4 GFLOPS。

Luo 等[5,16-17]先后面向 MT-3000 异构多核处理器提出了分布式并行密码恢复程序 MT-Office、SHA256 算法的并行实现 MT-SHA256 和高效异构并行密码恢复系统，通过对数据重排以支持向量化计算，通过异步 DMA 通信将 AM 划分为双缓冲区，实现计算和访存的重叠，针对 MT-3000 多级内存进行优化，合理利用 DDR4、AM 和向量寄存器之间的数据传输。与英特尔、飞腾平台相比，MT-Office 能够实现 60～218 倍的加速比，MT-SHA256 在单个加速核心上实现了 1045.68 MB/s 的最大吞吐量，比 MT-3000 上的 CPU 核心 C 代码实现快 9.84 倍，并行密码恢复系统相较完全运行在通用域上的系统实现了 18～53 倍的加速比。

Yin 等[18]面向 MT-3000 处理器进行代码相似性检测的加速，研究主要集中于矩阵乘法的优化，将 $\boldsymbol{A}$ 矩阵的数据载入 SM，$\boldsymbol{B}$ 和 $\boldsymbol{C}$ 矩阵的数据载入 AM，每次读出一个 A 的标量数据并广播为向量，与 B 中的一个向量数据进行乘法运算，最后累加写入 C 中。此外，将 AM 均等划分为四个缓冲区，利用 DMA 数据传输双缓冲方法，将 DMA 的数据传输时间隐藏到计算过程中。在应用前述两个访存优化方法后，相比未优化之前实现了 145 倍的加速比。

以上研究都从具体应用出发，分析关键数据的访存特点，通过数据重排优化顺序访存、SM/AM 均等划分形成多缓冲区、软件实现片上内存缓存管理、利用数据局部性进行数据预取等方法，有效提升了

程序的访存性能。然而,这些优化都是针对具体应用的定制化方案,且需要对底层硬件架构有较深理解。因此,针对 MT-3000 平台的低门槛、高普适性的访存优化方法亟待提出。

3　框架设计与实现

针对 MT-3000 众核架构访存优化面临的高门槛与低通用性挑战,本文面向 MT-3000 缓存 SM 显式软件管理,提出一种特征驱动缓存加速访存优化框架。该框架通过自动化的访存特征提取、策略推演和缓存管理,实现从访存特征到缓存配置的闭环优化流程。本框架优先选取 SM 作为缓存优化对象,核心考虑在于 SM 面向标量数据管理,其对应的标量计算逻辑简洁、数据访问模式相对规整,优化实现难度较低,能够快速搭建并验证“特征提取、策略推演、缓存管理”的完整框架流程,降低初期方案验证的复杂度;同时,SM 作为 MT-3000 内标量数据的核心存储载体,其访存效率对大量标量主导的计算任务性能影响显著,优化价值明显。

3.1　框架总体架构

本文提出的访存优化框架由三个核心模块组成,每个模块针对性地解决引言中提出的三个关键问题。

针对问题(1)访存特征表征粗粒度,本框架设计了访存特征提取模块 MACE-C。该模块通过静态分析、动态跟踪和特征提取三个阶段,将程序代码的访存行为转化为结构化的访存特征,实现了对程序中单个变量全生命周期访存行为的精确刻画,为后续优化提供细粒度、全面且结构化的访存特征信息。

针对问题(2)优化策略通用性不足,本框架构建了自适应缓存管理策略库 MT-cache。该策略库基于数据流特征分类,预设了批量缓存、单缓冲区和直接映射三种通用且可组合的优化模式,有效封装了底层并行编程模型的复杂实现细节。通过将不同特征的数据流映射到相应的优化策略中,框架实现了片上内存资源的智能分配与管理,大幅降低了开发人员的优化门槛。

针对问题(3)缺乏自动化优化框架,本框架提出了策略推演模型 LADWIM。该模型通过理论驱动的方法实现了从访存特征到缓存配置的自动化转换,构建了基于局部性感知和密度加权的数学模型,通过多变量联合优化,对各变量进行 SM 空间分配,确定最优缓存策略与缓存行大小、缓存行数量等参数配置。

如图 4 所示,该图直观呈现了框架的工作流程

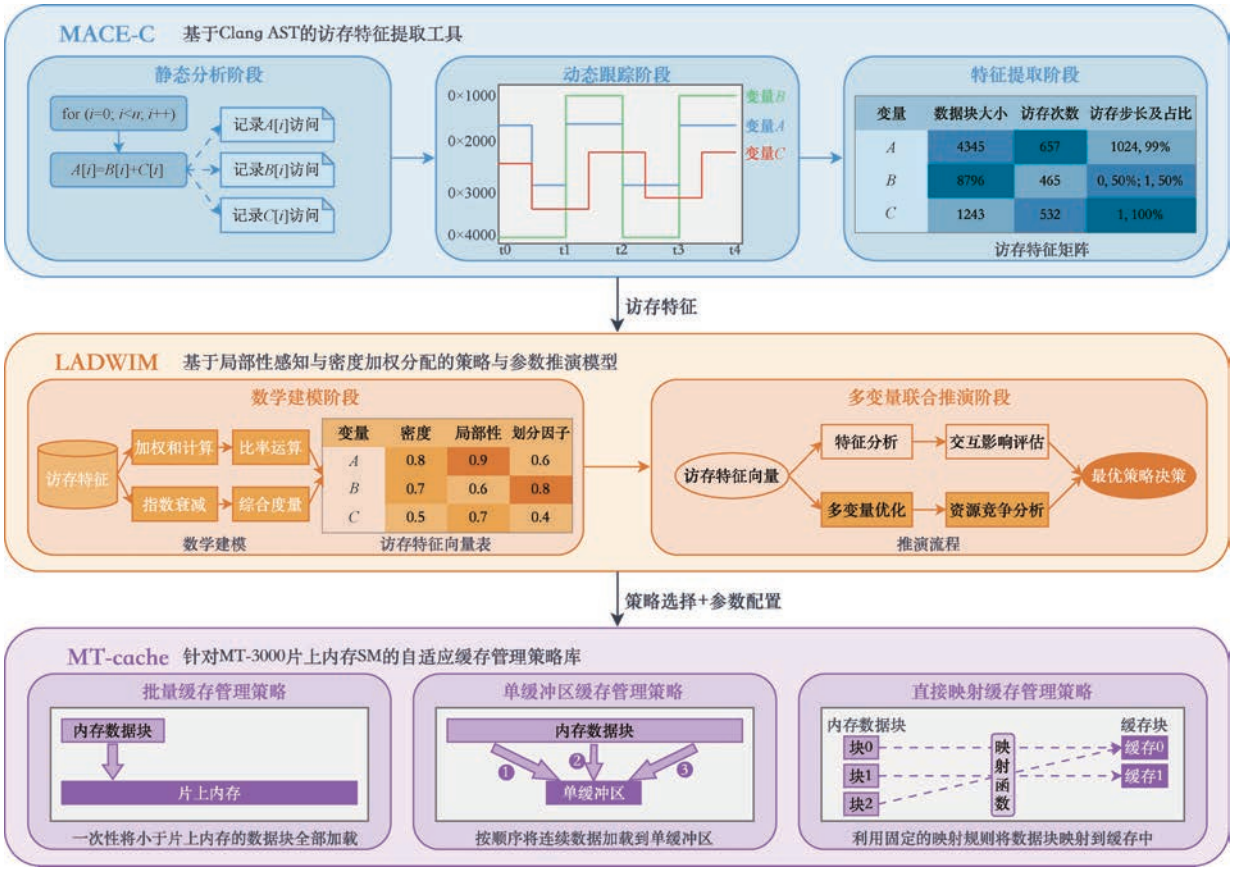

图 4　特征驱动缓存加速访存优化框架结构示意图

和组件交互关系，实现了从程序源码到优化配置的全自动化转换。MACE-C 模块通过静态分析、动态跟踪和特征提取三个阶段，将程序代码的访存行为转化为结构化的访存特征；LADWIM 模块基于这些特征进行数学建模和多变量联合优化推演，生成最优缓存管理策略决策和参数配置；MT-cache 模块根据推演结果，从批量缓存、单缓冲区和直接映射中选择合适的缓存管理策略。

3.2 访存特征提取模块 MACE-C

MACE-C[①] 采用动静结合的分析方法，基于 Clang/LLVM 19.0 工具链实现，主要面向 MT-3000 处理器架构的性能优化需求。

如图 5 所示，MACE-C 采用三阶段处理架构。静态分析阶段基于 Clang AST 技术对源代码进行分析和插桩；动态监测阶段通过插入的监测代码捕获实际运行时的访存信息；访存特征分析阶段对收集的数据进行处理，生成结构化的访存特征报告。

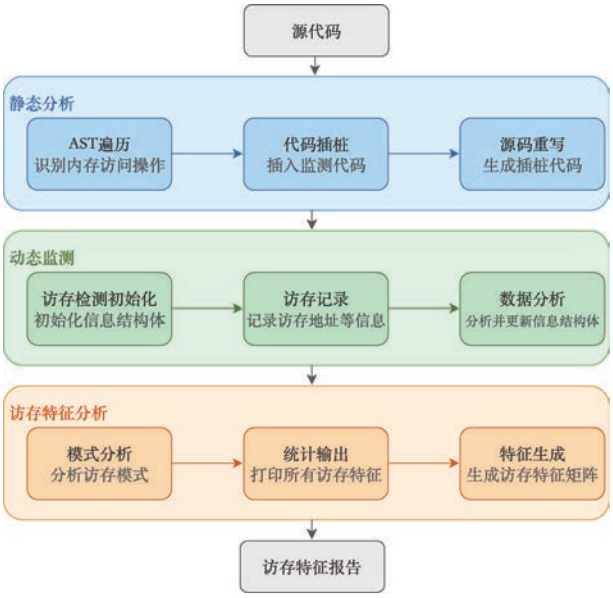

图 5 MACE-C 三阶段处理架构示意图

静态分析阶段的核心是源代码插桩算法，如算法 1 所示。该算法使用 Clang 构建抽象语法树，通过 RecursiveASTVisitor 模式遍历代码，识别访存操作并插入监测代码。相比于传统的二进制插桩技术，源代码级插桩能够保留更多程序语义信息，提供更精确的变量类型、作用域和函数层次关系等上下文。

算法 1. 静态代码分析和插桩算法

输入：源代码文件 S，目标函数集合 F_t

输出：插桩后源代码文件 S'

1. 使用 Clang 构建抽象语法树 T
2. 在适当位置插入内存分析器定义
3. FOR each 函数声明 $f \in T$ DO
4. IF $F_t = \emptyset$ OR $f.name \in F_t$ THEN
5. FOR each 参数或变量 $v \in f$ DO
6. IF MeetsInstrumentationCriteria(v) THEN
7. 插入内存分析器初始化代码
8. FOR each 访存点 $e \in f$ DO
9. 插入访问记录代码
10. 在函数返回点插入内存分析代码
11. 输出修改后的代码文件 S'

算法 1 涉及变量和符号的定义如表 1 所示。

表 1 算法 1 变量与符号定义

变量/符号	定义
S	源代码文件
F_t	目标插桩函数集合
S'	插桩后的源代码文件
T	抽象语法树(Abstract Syntax Tree)
f	函数声明节点
$f.name$	函数的名称
v	参数或变量声明节点
e	内存访问表达式节点

算法 1 的时间复杂度为 $O(n \times m)$，其中 n 为目标函数集合 F_t 中的函数数量，m 为单个函数中 AST 节点的平均数量。在实际应用中，AST 遍历采用深度优先策略，每个节点仅被访问一次，且插桩操作通过 Rewriter 类的增量修改机制实现，避免了重复的文件读写开销。对于典型的科学计算程序，单个函数的 AST 节点数量通常在 10^2 至 10^3 量级，整个插桩过程可在毫秒级完成。空间复杂度主要取决于 AST 的规模和 Rewriter 的修改缓冲区，为 $O(m)$。

该算法的核心是识别和处理三类关键的 AST 节点：变量声明节点、内存访问表达式节点和函数返回节点。对于变量声明节点，MACE-C 会对数组、指针和结构体类型的变量插入访存分析器初始化代码，记录变量名、所在函数、基地址和类型大小等信息。对于内存访问表达式节点，MACE-C 处理数组下标表达式、指针解引用操作和结构体成员访问表达式，在这些表达式后插入访存记录代码。对于函数返回节点，MACE-C 在函数返回前插入访存分析和结果报告代码，生成访存统计信息。插桩过程使用 Clang 的 Rewriter 类进行源代码修改，能够保持原始代码结构，通过 SourceLocation 接口进行精确

① 相关代码已开源，项目网址：https://github.com/Forgoys/MemProfMT。

定位,确保代码正确插入。

动态监测阶段的核心是访存记录算法,如算法 2 所示。

算法 2. 访存记录算法

输入:内存分析器结构体 *prof*,访存地址 *curr_addr*

输出:更新后的 *prof*

1. *prof.base_addr*←min(*prof.base_addr*,*curr_addr*)
2. *prof.end_addr*←max(*prof.end_addr*,*curr_addr*)
3. *prof.total_accesses*←*prof.total_accesses*+1
4. *step*←|*curr_addr*-*prof.last_addr*|/*prof.type_size*
5. IF *step*<65536 THEN
6. 更新步长统计数组 *prof.patterns* 和 *prof.pattern_counts*
7. *prof.last_addr*←*curr_addr*

算法 2 涉及变量和符号的定义如表 2 所示。

表 2　算法 2 变量与符号定义

变量/符号	定义
prof	内存分析器结构体
curr_addr	当前访存地址
prof.last_addr	上一次访存地址
prof.base_addr	访存范围的起始地址
prof.end_addr	访存范围的结束地址
prof.total_accesses	累计的总访存次数
prof.type_size	变量的类型大小
step	当前访存步长
prof.patterns	访存步长存储数组
prof.pattern_counts	步长出现次数存储数组

算法 2 的时间复杂度为 $O(k)$,其中 k 为最大记录的访存模式数量。对于每次访存操作,算法首先执行常数次地址比较和更新操作,然后在 *patterns* 数组中查找当前步长,最坏情况下需要遍历整个数组。空间复杂度为 $O(k)$,主要用于存储 *patterns* 数组和 *pattern_counts* 数组。

该算法的核心是根据本次访存的地址更新 *prof*。在获取到访存地址 *addr* 后,首先更新当前的访存地址与变量访问范围的首尾地址,同时记录下一次访存触发动态检测后计算步长所需的前序访存地址,然后累加总访存次数,计算当前的访存步长 *step*。访存步长计算采用地址差值归一化处理,计算公式如公式(1)所示。

$$step = \frac{|\ addr_{\text{curr}} - addr_{\text{prev}}\ |}{type_{\text{size}}} \tag{1}$$

其中,$addr_{\text{curr}}$ 代表当前访存地址,$addr_{\text{curr}}$ 代表前一次访存地址,$type_{\text{size}}$ 代表变量类型的字节大小。这种归一化使得不同数据类型的访问模式可以在同一标准下比较。

访存模式的记录使用固定大小的数组 *patterns* 和 *pattern_counts* 分别存储不同的步长值及其出现次数。当遇到新的步长时,将其添加到数组中;当遇到已有步长时,增加相应的计数。这种设计虽然限制了记录的最大模式数量为 16 种,但实际应用中这已足够覆盖大多数程序的主要访存特征。同时,MACE-C 对大于等于 65536 的超大步长进行了特殊处理,避免异常值对统计结果的干扰,能够有效识别 MT-3000 处理器中常见的连续访问、跨步访问和随机访问等多种模式。

访存特征分析阶段对收集的原始数据进行处理,提取变量大小、总访问次数、主要访存步长及其占比等关键特征。该阶段采用排序算法对访存步长按出现频率排序,设置频率阈值过滤噪声数据,只有超过 5%占比的访存模式才会在最终报告中显示,确保分析结果聚焦于对性能有显著影响的主要访存模式。

3.3　策略推演模型 LADWIM

LADWIM① 模型是框架的核心决策组件,通过数学建模将访存特征转化为缓存管理策略和缓存行大小、缓存行数量等参数配置。该模型设计为两个阶段:访存特征数学建模阶段和多变量联合推演阶段。

3.3.1　访存特征建模

访存特征建模的核心是将复杂的访存行为转化为可量化的特征指标。LADWIM 首先构建结构化的访存特征向量,定义如公式(2)所示。

$$\boldsymbol{F}_{a_i} = (S_{a_i}, N_{a_i}, \{(s^j_{a_i}, p^j_{a_i}) \mid j \in \{1,2,\cdots,m_i\}\} \\ D_{a_i}, L_{a_i}) \tag{2}$$

其中,i 为变量索引,表示第 i 个变量;j 为步长索引,表示第 j 种访存步长模式;a_i 表示第 i 个变量的访存次数;s^j 表示第 j 种访存步长;p^j 表示第 j 种步长的出现占比;S_{a_i} 为变量 a_i 的访存空间大小;N_{a_i} 为访存次数;D_{a_i} 为访存密度;L_{a_i} 为空间局部性。

不同于传统的单一指标表征,该特征向量综合考虑了变量的空间大小、访问频率、访问模式以及局部性等多维特征,形成了对变量访存行为的全面描述。

访存密度定义为单位访存空间上的访问次数,反映了访存操作的集中程度,定义如公式(3)所示。访存密度越高,表明该变量在单位空间内被访问的频率越高,从缓存中受益的可能性也越大。

① 相关代码已开源,项目网址:https://github.com/Forgoys/MASAMT。

$$D_{a_i}=\frac{N_{a_i}}{S_{a_i}} \tag{3}$$

空间局部性通过步长分布的加权求和计算，能够准确反映数据在空间维度上的聚集程度，定义如公式(4)所示。指数函数 $e^{-s^j_{a_i}}$ 确保步长增大时局部性指标呈非线性下降，更符合实际缓存性能特性。该公式体现了步长越小，空间局部性越高的基本原理。

这种设计基于以下考虑：步长为 0 的重复访问具有最高的局部性权重(权重为 1)，步长为 1 的连续访问具有很高的局部性权重(权重约为 0.37)，随着步长增大，局部性权重快速衰减，体现了缓存友好性的递减特性。这种非线性衰减模式与实际的缓存性能特征相符，当访存步长超过缓存行大小时，缓存效率会显著下降。

$$L_{a_i}=\sum_{j=1}^{m_i}(p^j_{a_i}\times e^{-s^j_{a_i}}) \tag{4}$$

空间划分因子 f_{a_i} 综合考虑局部性和密度，作为变量对缓存资源需求强度的综合表征，定义如公式(5)所示。空间划分因子越大，表明变量对缓存空间的需求越高，应分配更多的 SM 空间资源以提高整体性能。

$$f_{a_i}=L_{a_i}\times D_{a_i} \tag{5}$$

3.3.2 多变量联合优化

多变量联合优化阶段实现了从访存特征到缓存策略及其参数的自动化推导。LADWIM 采用迭代优化机制进行缓存资源分配和策略推演，如图 6 所示。

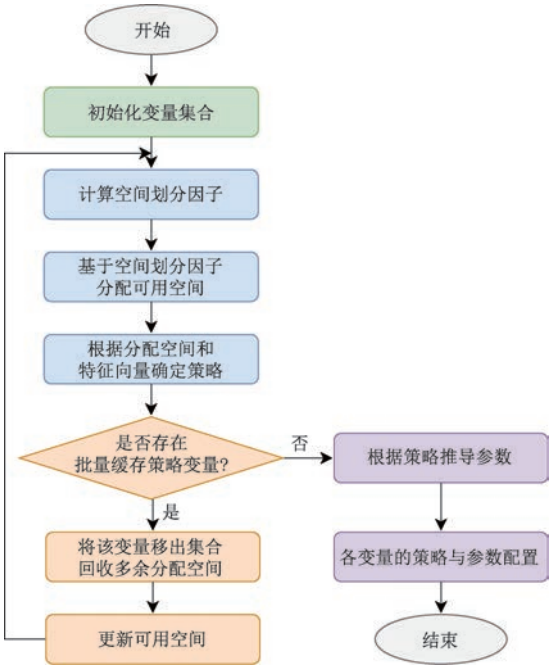

图 6 LADWIM 多变量联合推演流程图

该阶段首先需要解决 SM 空间资源分配问题，然后进行策略选择与参数配置推演。

资源分配采用基于空间划分因子的比例分配策略，如算法 3 所示。该算法通过迭代优化机制实现 SM 空间的智能分配，在满足可完全缓存变量需求的同时，为其他变量按需分配资源。

算法 3. 缓存资源分配算法

输入：待优化变量集合 ν，可用 SM 空间大小 C_{total}

输出：更新后的变量集合 ν

1. $\nu_{\text{remain}}\leftarrow\nu, C_{\text{remain}}\leftarrow C_{\text{total}}$
2. WHILE $\nu_{\text{remain}}\neq\varnothing$ DO
3. FOR each $v\in\nu_{\text{remain}}$ DO
4. $v.C\leftarrow C_{\text{remain}}\times\dfrac{v.f}{\sum_{u\in\nu_{\text{remain}}}u.f}$
5. $\nu_{\text{bulk}}\leftarrow\{v\in\nu_{\text{remain}}:v.C\geqslant v.S\}$
6. $\nu_{\text{unsuitable}}\leftarrow\{v\in\nu_{\text{remain}}:v.f=0\}$
7. IF $\nu_{\text{bulk}}\cup\nu_{\text{unsuitable}}=\varnothing$ THEN BREAK
8. 设置 ν_{bulk} 中变量策略为批量缓存，$\nu_{\text{unsuitable}}$ 中变量策略为不适用
9. $C_{\text{remain}}\leftarrow C_{\text{remain}}-\sum_{v\in\nu_{\text{bulk}}}v.S$
10. $\nu_{\text{remain}}\leftarrow\nu_{\text{remain}}\setminus(\nu_{\text{bulk}}\cup\nu_{\text{unsuitable}})$
11. IF $\nu_{\text{remain}}\neq\varnothing$ THEN
12. FOR each $v\in\nu_{\text{remain}}$ DO
13. $v.C\leftarrow C_{\text{remain}}\times\dfrac{v.f}{\sum_{u\in\nu_{\text{remain}}}u.f}$
14. 返回 ν

算法 3 涉及变量和符号的定义如表 3 所示。

表 3 算法 3 变量与符号定义

变量/符号	定义
ν	所有待优化变量的集合
ν_{remain}	未确定资源分配的变量集合
C_{total}	SM 的总可用空间大小
C_{remain}	当前剩余的可用 SM 空间大小
v	单个变量
$v.C$	变量 v 已分配的 SM 空间大小
$v.s$	变量 v 的访存空间大小
$v.f$	变量 v 的空间划分因子
ν_{bulk}	适用批量缓存的变量集合
$\nu_{\text{unsuitable}}$	不适用缓存策略的变量集合

算法 3 时间复杂度为 $O(k\times n)$，其中 k 为迭代次数，n 为变量数量，迭代次数最大不超过变量数量。在每轮迭代中，算法需要遍历 ν_{remain} 集合中的所

有变量进行资源分配,然后识别 ν_{bulk} 和 $\nu_{\text{unsuitable}}$ 集合,并更新剩余资源。空间复杂度为 $O(n)$,主要用于存储变量集合和相关属性。

该算法采用迭代优化方法对 SM 空间进行分配。每轮迭代中,首先按空间划分因子 f 的比例将可用空间 C_{remain} 分配给所有待分配变量(第 3～4 行);然后识别分配空间超过自身大小的变量 ν_{bulk}(满足 $v.C \geqslant v.S$)和不适合缓存的变量 $\nu_{\text{unsuitable}}$(满足 $v.f=0$),将其策略确定并从待分配集合中移除(第 5～10 行);回收这些变量占用的空间后进入下一轮迭代。循环结束后,对剩余变量进行最终分配(第 11～13 行)。这种策略优先满足可完全缓存的变量,然后将剩余资源按比例分配给其他变量,确保高局部性、高访存密度的变量获得更多缓存资源。

完成空间分配后,需要根据变量特征和分配资源确定具体的缓存策略及其参数。策略选择基于变量特征和分配资源确定。对于分配空间 $C_{a_i} \geqslant S_{a_i}$ 的变量,采用批量缓存策略;对于 $C_{a_i} < S_{a_i}$ 但 $L_{a_i} > \lambda$ 的变量,采用单缓冲区策略;其他变量采用直接映射策略。特别地,对于只有步长为 0 的访问模式且局部性指标 L_{a_i} 小于 0.9 的变量,尽管局部性计算值可能较高,但实际上表现为随机访问,因此会采用直接映射策略而非单缓冲区策略。

参数推导针对不同策略计算其 set、line 参数最优取值配置。对于批量缓存策略,set 参数无实际意义,line 参数代表缓冲区大小为 line 字节。参数推导如公式(6)所示。

$$set_{a_i}=0, line_{a_i}=S_{a_i} \tag{6}$$

其中,$line_{a_i}$ 直接设置为变量 a_i 的访存空间大小 S_{a_i},实现一次性将所有数据加载到缓存中。

对于单缓冲区策略,set 参数无实际意义,关键参数是 line,表示缓冲区的大小设置为 2^{line} 字节,如公式(7)所示。

$$set_{a_i}=0, line_{a_i}=\lfloor \log_2(C_{a_i}) \rfloor \tag{7}$$

该策略通过将分配空间 C_{a_i} 近似为 2 的幂次值来确定缓存行大小,既满足硬件对齐要求,又最大化利用分配资源。

对于直接映射策略,需要同时确定 line 和 set 两个关键参数,分别表示缓存行数量为 2^{set},缓存行大小为 2^{line} 字节。计算方式如公式(8)和公式(9)所示。

$$line_{a_i}=\begin{cases}\max(4,\min(\alpha_{a_i},\beta_{a_i})), & \text{若 } s_{\max}>0\\ \lfloor \log_2(D_{a_i}) \rfloor, & \text{若 } s_{\max}=0\end{cases} \tag{8}$$

$$set_{a_i}=\left\lfloor \log_2\left(\max\left(1,\left\lfloor \frac{C_{a_i}}{2^{line_{a_i}}} \right\rfloor\right)\right) \right\rfloor \tag{9}$$

其中,相关辅助参数的计算如公式(10)、公式(11)和公式(12)所示。

$$s_{\max}=\max_{j\in\{1,2,\cdots,m_i\}} s_{a_i}^{j} \tag{10}$$

$$\alpha_{a_i}=\lceil \log_2(s_{\max}) \rceil+2 \tag{11}$$

$$\beta_{a_i}=\lfloor \log_2(C_{a_i}) \rfloor \tag{12}$$

其中,$s_{\max}$ 为变量的最大步长。

$line_{a_i}$ 参数基于变量的访存特性确定:当存在有效步长时,通过 α_{a_i} 和 β_{a_i} 的最小值来平衡局部性利用和资源约束;当全是随机访问时,基于访存密度确定缓存行大小。set_{a_i} 参数基于分配空间和 $line_{a_i}$ 值计算,确保在资源约束下最大化缓存行数量。

3.4　自适应缓存管理策略库 MT-cache

MT-3000 的 SM 缓存采用软件显式管理方式,不同于传统硬件缓存的自动管理机制。虽然组相联映射是硬件缓存的常用策略,但其依赖硬件的标签匹配和复杂的替换逻辑,在软件管理模式下实现会引入显著开销。包括标签查找、LRU 维护等逻辑,会抵消缓存带来的性能收益。因此,本文设计了三种适合 MT-3000 架构特点的软件管理策略:批量缓存管理策略将数据一次性读入 SM,适用于数据量小且访问密集的场景;单缓冲区缓存管理策略将数据分批读入 SM,适用于具有良好局部性的顺序访问;直接映射缓存管理策略通过灵活的 set 和 line 参数提供可调的空间划分能力,适合局部性较差的跨步访问。

MT-cache[①] 提供三种互补的缓存管理策略,采用基于 C 语言宏的统一接口设计,通过写回式缓存更新机制保证数据一致性。该策略库的设计目标是提供一套易于集成、无需特殊硬件支持且具有灵活适应性的软件级缓存管理方法,本文所提出的优化框架可根据不同变量的访存特征,自适应地选择合适的缓存管理策略,每个变量的缓存策略在访存策略分析阶段就已确定,不会涉及运行时策略切换的性能开销。

3.4.1　缓存组织结构

如图 7 所示,三种策略采用不同的缓存组织结构,以适应不同的访存模式和应用场景。

批量缓存策略针对高访问密度的小块数据进行优化,通过一次性加载和回写实现高效的数据管理。单缓冲区策略采用最简化的缓存组织结构,仅维护一个缓冲区,极大地简化了缓存管理的复杂性。直

① 相关代码已开源,项目网址:https://github.com/Forgoys/MT-Cache。

接映射策略实现了一种轻量级的缓存分区机制，将内存地址空间映射到多个缓存行中，在缓存容量和命中率之间取得平衡。

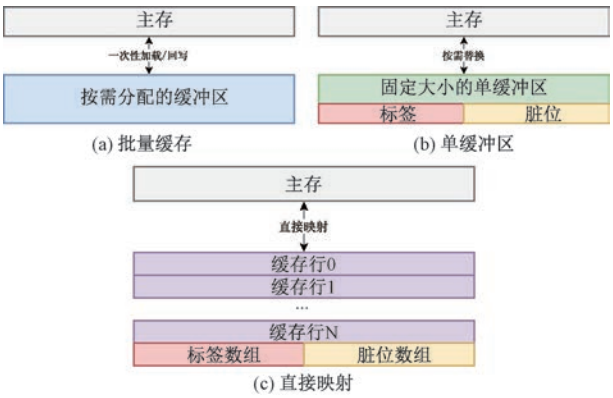

图 7 MT-cache 缓存管理策略的组织结构示意图

3.4.2 策略实现

批量缓存策略针对高访问密度的小块数据进行优化，通过一次性加载和回写实现高效的数据管理。该策略的核心优势在于避免了频繁的主存访问，大幅提升了数据访问性能。初始化时首先确定缓存大小并进行 64 位对齐机制，通过向上和向下对齐计算确保访存的高效性，同时避免 SM 与主存间进行 DMA 通信时的错误。读写接口相对简单，因为所有数据已经一次性加载到 SM 中，读取操作直接访问 SM 缓存中相应位置的数据，写入操作则直接修改 SM 缓存中的数据。缓存刷新时一次性将整个缓存块写回主存，这种批量写回策略减少了频繁的主存写入操作。

单缓冲区策略采用最简化的缓存组织结构，仅维护一个缓冲区，极大地简化了缓存管理的复杂性，减少了资源占用和额外计算开销。该策略通过维护当前缓存块的标签和脏位信息，实现高效的缓存替换和一致性管理。地址计算采用位运算优化，标签计算使用右移操作 $T=A>>L$，偏移量计算使用位与操作 $O=A\&(2^L-1)$，避免了开销较大的除法运算。核心算法包括缓存命中判断和缓存替换逻辑，未命中时会触发缓存替换，首先检查当前缓存行的脏位状态，如果脏位为 1 则将缓存行数据写回主存，然后从新地址加载数据到缓存并更新标签。

直接映射策略实现了一种轻量级的缓存分区机制，将内存地址空间映射到多个缓存行中，在缓存容量和命中率之间取得平衡。该策略特别适合访存模式规律性不强或数据局部性不理想的场景，可以在数据局部性较差的情况下利用 SM 的高带宽低延迟特性进行访存加速。地址分解采用位运算实现，将待访问的主存地址 A 分解为标签 T、索引 I 和偏移量 O 三个部分，通过映射函数确定缓存行位置。该策略维护标签数组和脏位数组管理缓存状态，初始化时所有标签被设置为无效值，所有脏位被清零。缓存替换机制通过检查标签匹配来判断缓存命中，未命中时根据脏位状态决定是否需要写回数据。

需说明的是，本文提出的特征驱动优化方法同样适用于 AM 缓存优化。MACE-C 获取到适合向量化计算的变量的访存特征，例如，某变量的访存步长占比为 1，根据该类访存特征，定位可向量化的变量，进而利用 AM 缓存进行优化；LADWIM 的策略推演逻辑无需本质调整，可直接映射至 AM 的缓存配置需求。二者的核心差异在于，AM 面向向量数据存储，其优化需先将原始代码中的标量计算逻辑转化为向量化计算，完成向量化适配后，即可复用本框架的缓存管理策略与推演模型实现 AM 访存优化。

3.4.3 数据一致性说明

MT-cache 策略库基于 MT-3000 的架构特性设计数据管理机制。由于 SM 是控制核私有的片上存储，每个控制核拥有独立的 64KB SM 空间，因此不存在多核心竞争访问同一 SM 的场景，从硬件层面避免了传统的缓存一致性问题。

本框架采用软件显式管理的方式处理数据更新。所有策略均使用写回式缓存更新机制，通过统一的缓存刷新接口将修改后的数据写回主存。这种显式的数据同步由程序员通过 API 调用控制，确保在数据共享点之前完成必要的写回操作。

对于跨控制核的数据共享场景，数据一致性由上层编程模型保证。MT-3000 的 hthreads 编程模型提供了同步机制，应用程序通过合理的任务划分和同步机制确保不同控制核访问共享数据的正确性。这种一致性保证属于编程模型层面，超出了本缓存管理框架的研究范畴。

需要说明的是，本框架当前聚焦于控制核私有的 SM 优化。如果未来扩展到 GSM、HBSM 等共享存储层次，则需要考虑更复杂的多核协同访问和一致性维护机制，这是后续研究的重要方向。

3.5 框架工作流程与实现

3.5.1 框架工作流程

完整的优化流程实现了从程序源码到优化配置的全自动化转换，分为特征获取、策略推演和代码优化三个主要阶段，如图 8 所示。

特征获取阶段：首先通过 MACE-C 对源代码进行分析与插桩处理，在这一阶段基于抽象语法树对

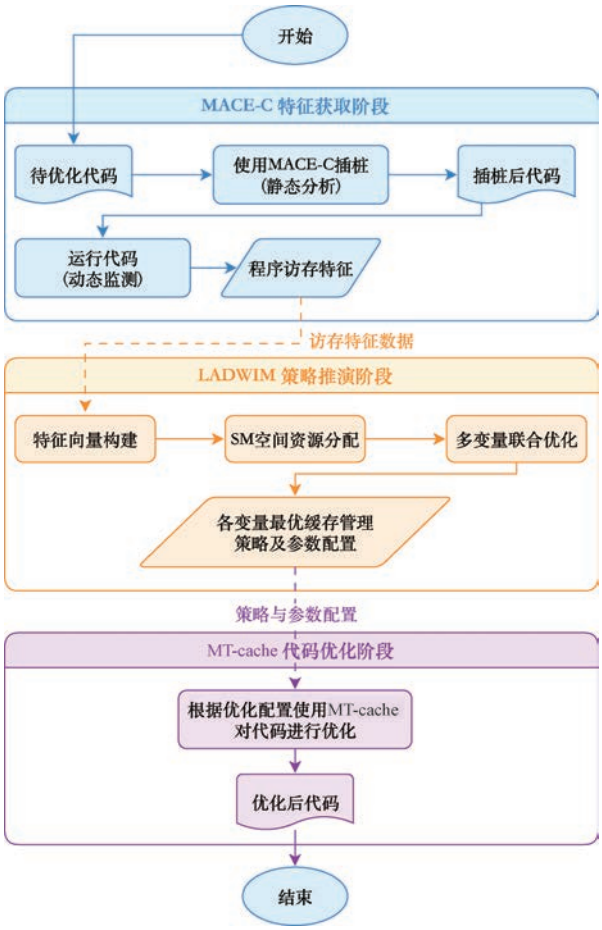

图 8　框架工作流程图

源代码进行静态分析,识别访存点并在合适位置插入轻量级监测代码。插桩后的代码结构与原始代码保持一致,但增加了访存记录与分析功能。编译并运行代码以收集实际执行时的访存特征,插入的监测代码会捕获每次访存的地址信息,计算访存步长分布,并记录每个变量的访存模式。程序执行结束时,MACE-C 输出结构化的访存特征报告,包括各变量的访存空间大小、总访问次数及主要访存步长等关键信息。

策略推演阶段:LADWIM 模型接收访存特征数据并进行深入分析,推导出每个变量的最优缓存管理策略及参数配置。此阶段首先将原始访存特征转换为结构化的访存特征向量,计算空间局部性和访存密度等关键指标,构建准确量化数据局部性的数学模型。随后执行多变量联合优化,通过迭代分配算法为各变量分配合适的 SM 缓存空间,该分配过程考虑变量间的互相影响,优先满足适合批量缓存的变量,并将剩余资源按空间划分因子的比例分配给其他变量。SM 空间分配完成后,模型根据变量的局部性特征和分配到的 SM 空间资源,从三种策略中选择最适合各变量的缓存管理策略,并推导出相应的参数配置。

代码优化阶段:开发人员使用 MT-cache 缓存管理策略库根据 LADWIM 的推演结果对代码进行优化实现。MT-cache 通过统一的宏接口封装了底层缓存管理的复杂细节,在这一阶段开发人员根据每个变量的推荐策略和参数配置改写代码为相应的缓存管理代码,包括缓存初始化、读写操作和缓存刷新等功能。这些代码可以无缝集成到原始程序中,通过高效利用 MT-3000 的 SM 片上内存,显著提升访存性能。优化后的代码保持了原始程序的功能逻辑,但具备了更高效的访存能力。

这种特征驱动的设计理念通过三个模块的协同工作,形成了完整的优化闭环。框架具有良好的扩展性和适应性,随着硬件平台的变化,可以调整 LADWIM 模型中的策略决策算法或 MT-cache 中的缓存实现细节,而无需改变整体框架架构,确保了系统在面对多样化场景时的稳定性和灵活性。

3.5.2　框架实现

框架在实现过程中采用了多项关键技术,确保了实用性和高效性。

(1)源码级插桩技术:相比二进制插桩,源码级插桩能够保留更多程序语义信息,提供精确的变量类型、作用域和函数层次关系。通过 Clang AST 的完整语法树模型,框架不仅能识别显式的数组访问,还能捕获指针解引用和结构体成员访问中隐含的内存操作。

(2)低开销动态监测:动态监测代码采用高度优化的内联函数实现,访存记录采用固定大小数组进行统计,在空间效率和表达能力之间取得平衡。虽然监测阶段会引入一定开销,但这一成本是一次性的,且远低于优化带来的长期收益。

(3)迭代优化的资源分配:LADWIM 采用迭代方法进行 SM 空间分配,每轮迭代识别并处理可完全缓存和不适用缓存的两类特殊变量,确保资源向高价值变量倾斜。算法的时间复杂度接近线性,能够高效处理多变量优化场景。

(4)统一的策略接口:MT-cache 策略库通过模板化设计和统一接口,屏蔽了不同缓存策略的实现细节。用户只需根据 LADWIM 的推荐选择策略类型和配置参数,无需深入了解底层的 API 调用和缓存替换逻辑,显著降低了使用门槛。

3.6　框架的架构适应性分析

本文提出的特征驱动缓存加速访存优化框架的

核心设计理念基于软件可管理的片上内存机制，这一机制并非 MT-3000 处理器所独有，而是众核处理器架构中一种重要的内存管理方式。因此，本框架具有良好的架构适应性，可扩展应用于具有类似内存层次结构的众核处理器平台。

软件可管理片上内存是一种由程序员或编译器显式控制的快速片上存储空间，通常具有以下特征：位于处理器片上，访问延迟低、带宽高；需要通过特定 API 或 DMA 机制进行数据搬运；容量有限，需要精细化管理以提高利用率；不具备硬件缓存一致性机制，数据管理完全由软件控制。

本框架的设计方法和优化策略可适用于多种具有软件可管理片上内存特征的处理器架构，例如神威 SW26010 处理器[19]，每个从核配备 64 KB 的局部数据存储器，需要程序员通过 DMA 机制显式地在主存和 LDM 之间搬运数据，具有类似特征的还有神威 SW26010Pro 处理器[20]、Cell Broadband Engine[21]等。这些架构的共同特点是将缓存管理权交由软件，通过精细化的数据编排来弥补访存带宽的不足。本框架提出的访存特征量化方法、多策略缓存管理库和自动化推演模型，为这类架构提供了系统化的优化方案。

将本框架应用于其他具有软件可管理片上内存的处理器架构时，主要需要适配以下三个方面：

(1)访存特征提取模块适配：MACE-C 基于 Clang AST 实现，具有良好的源码级可移植性。对于不同架构，只需修改动态监测阶段的线程标识和输出函数接口，即可适配不同的运行环境。

(2)缓存管理策略库适配：MT-cache 策略库封装了底层的数据搬运和缓存管理逻辑。对于不同架构，需要将底层的 scalar_load/scalar_store API 替换为目标平台的数据搬运接口，策略的逻辑结构和参数配置方式保持不变，确保了策略库的跨平台可移植性。

(3)策略推演模型适配：LADWIM 模型基于访存特征向量进行推演，其数学模型和推演算法与具体硬件平台无关。

综上所述，本框架的设计充分考虑了众核处理器架构的共性特征，通过模块化设计实现了核心算法与底层硬件接口的解耦。对于具有软件可管理片上内存的处理器架构，本框架能够以较低的移植成本提供有效的访存优化支持，为众核处理器的应用优化提供了通用性强、可扩展的系统化解决方案。

4 实验与分析

为了验证框架的有效性，本节采用递进式验证策略：首先通过基础性能对比实验验证缓存管理策略的实用性；然后通过正确性验证实验确认策略选择机制的准确性；接着通过参数推演验证实验评估 LADWIM 模型的精度；最后通过综合性能评估全面展示框架的优化效果。这种验证策略确保了从单个组件到整体框架的系统性评估，为框架的实际应用提供可靠的实验依据。

4.1 实验环境与设置

4.1.1 实验环境配置

实验在天河新一代超级计算机系统的 TH-3M1 集群上进行，环境配置如表 4 所示。

表 4 实验环境配置

名称	参数
CPU	Aarch64 架构，16 核
DSP	MT-3000 加速簇，单节点 4 个
内存	32GB(单加速簇)
操作系统	Ubuntu 20.04.2 LTS
编译器	gcc 9.3.0，MT-3000-gcc 8.3.0

4.1.2 基准测试集与应用程序

实验采用 PolyBenchMT① 基准测试集和航空发动机内流高效数值模拟平台应用 X-Blue 进行评估。PolyBenchMT 是基于 PolyBench/GPU 迁移到 MT-3000 平台的版本，包含线性代数、数据挖掘、模板计算和卷积等典型计算模式的 22 个算子。X-Blue 是面向我国超算系统的航空发动机内流数值模拟应用，采用分层几何区域分解方法，在万核规模计算任务中能实现超过 90%的并行效率[22]。

4.2 策略选择与参数推演有效性验证

为验证 LADWIM 模型的有效性，本节设计了策略选择正确性验证和参数推演准确性验证两个层次的实验。表 5 展示了验证实验中使用的关键变量及其访存特征，这些变量涵盖了四种不同的缓存策略场景，为全面评估模型的推演能力提供了代表性样本。

表 5 中的变量展现了不同的访存模式特征：SYRK 算子的 a 变量表现为随机访存，无明显规律性步长；c 变量在 MINI 负载下具有规律的 0 步长和 1 步长访存，且数据量较小；ADI 算子的 A、B、X

① 相关代码已开源，项目网址：https://github.com/Forgoys/PolyBenchMT。

三个变量均表现出良好的空间局部性,主要以 0 步长和 1 步长访存为主;JACOBI2D 算子的 A 变量则呈现复杂的混合访存模式,包含小步长的局部访存和大步长的跨行访存。这些不同的访存特征为验证 LADWIM 模型在各种场景下的推演能力提供了理想的测试用例。

4.2.1　策略选择一致性验证

本实验通过将 LADWIM 模型的推演结果与基于访存特征的理论分析进行对比,验证策略选择的合理性。我们选择了四种典型场景进行验证:缓存不适用、批量缓存、单缓冲区和直接映射策略。

缓存不适用情况验证:选择 SYRK 算子 syrk_kernel 核函数中的 a 变量进行验证。以 STANDARD 负载为例,如表 5 所示,该变量访存位置分散,空间局部性较差,模型推演结果为访存策略不适用。实验将未使用策略优化的 DSP 执行时间与使用直接映射策略优化后的 DSP 执行时间进行对比验证,结果如图 9(a)所示。

表 5　验证实验中关键变量的访存特征及推演结果

算子	变量	负载	空间大小(字节)	访问次数	主要访存步长及占比	推演策略	line	set
SYRK	a	STANDARD	8388608	2147483648	—	缓存不适用	0	0
	c	MINI	22528	131072	0(50.0%);1(50.0%)	批量缓存	22528	0
ADI	A	STANDARD	352248	3142656	0(66.7%);1(33.3%)	单缓冲区	14	0
	B	STANDARD	352256	4190208	1(75.0%);0(25.0%)	单缓冲区	14	0
	X	STANDARD	352256	3142656	0(33.3%);1(33.3%);2(33.3%)	单缓冲区	13	0
JACOBI2D	A	STANDARD	184312	104448400	1(20.0%);2(20.0%);1023(20.0%)	直接映射	5	10

从图 9(a)可以看出,在 DSP 上优化前后的执行时间几乎一致,加速比平均为 1.0,且相对于 CPU 并无加速效果,验证了模型推演结果的正确性。

批量缓存策略验证:选择 SYRK 算子 syrk_kernel 核函数中的 c 变量进行验证。该变量只在 MINI 负载下适用于批量缓存策略。实验分别对 c 变量使用三种策略优化,同时与 CPU 串行执行时间和未优化的 DSP 执行时间进行对比,结果如图 9(b)所示。

从图 9(b)可以看出,三种缓存管理策略优化后的执行时间一致,但批量缓存策略可以在占用 SM 空间更小(22 KB)的前提下,取得和另外两种策略相同的加速效果,验证了模型推演出的批量缓存策略是正确的。

单缓冲区策略验证:选择 ADI 算子 adi_kernel1 核函数中的 A、B、X 变量进行验证。以 STANDARD 负载为例,如表 5 所示,这些变量的步长 0 和 1 的占比很大,适合单缓冲区策略。实验将三个变量分别使用单缓冲区和直接映射策略进行优化,结果如图 9(c)所示。

从图 9(c)可以看出,使用单缓冲区策略的核函数相对于使用直接映射策略和不使用策略的加速比远大于 1,随着计算负载近似线性增加,最高可达 29.57 倍,相较 CPU 串行执行有明显性能提升,充分证明了模型推演结果的正确性。单缓冲区策略相对直接映射策略的性能优势主要源于单缓冲区管理策略简单的管理逻辑和较小的计算开销,如表 5 所示的 MACE-C 提取的访存特征,ADI 算子的 A、B、X 三个变量主要访存步长为 0 和 1,占比分别达到 66.7%/33.3%、75%/25% 和 33.3%/33.3%。单缓冲区策略采用较大的缓存行(line=13-14,即 8KB-16KB),能够完整容纳连续访问的数据块。对于步长为 0 的重复访存和步长为 1 的连续访存模式,缓存行内的数据可被充分复用,变量 A 和 B 的理论缓存命中率可达 95%以上。相比之下,直接映射策略将 SM 空间划分为多个较小的缓存行,在此类访存模式下,相较于单缓冲区管理策略会引入更多的缓存行替换,导致大量额外计算开销,使得最终性能优化效果显著弱于单缓冲区管理策略。

直接映射策略验证:选择 JACOBI2D 算子 jacobi_kernel1 核函数中的 A 变量,在 SMALL、STANDARD 和 LARGE 三个计算负载下验证。以 STANDARD 负载为例,如表 5 所示,该变量呈现出复杂的访存模式:存在占比达 20%的大步长(1023),同时伴有各占 20%的步长 0 和步长 1 的访问模式。这种混合访存模式表明该变量在进行跨行访问(大步长)的同时,存在对相邻元素的顺序访问(步长 1)和对同一位置的重复访问(步长 0),正是适合直接映射策略的典型场景。实验将 A 变量分别使用单缓冲区和直接映射策略进行优化,直接映射策略参数按照模型推演结果设置为(set=10,line=5),单缓冲区策略参数设置为 line=10 以匹配数据访问特征,结果如图 9(d)所示。从图中可以看出,使用直接映射策略的核函数相对于使用单缓冲区策略和未优化版本均有明显的性能提升,加速比远大于 1,且随着计算

负载增加呈现稳定的优化效果，充分验证了模型推演的正确性。

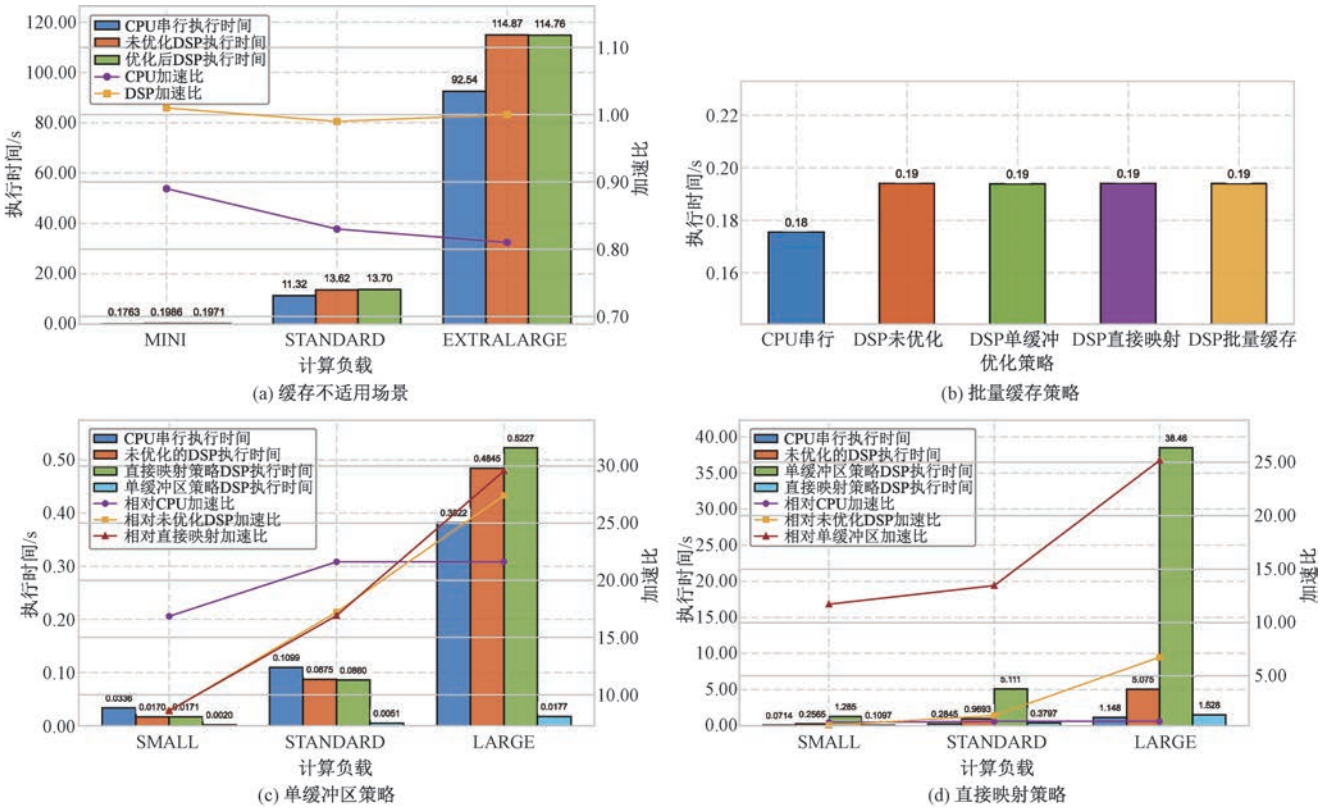

图 9 策略选择验证结果

4.2.2 参数推演准确性验证

为验证 LADWIM 模型参数推演的准确性，实验对单缓冲区和直接映射两种策略的参数配置进行验证。

单缓冲区策略参数验证：选择 ADI 算子 adi_kernel1 核函数中的 A、B、X 三个变量，在 STANDARD 负载下进行验证。如表 5 所示，LADWIM 推演出的最优参数组合为 A 变量取 14、B 变量取 14、X 变量取 13。实验构建了全面的参数搜索空间，每个参数取值范围为 7 至 15，在排除不满足约束条件的组合后，实际测试了 701 种有效配置，实验结果如图 10 所示。

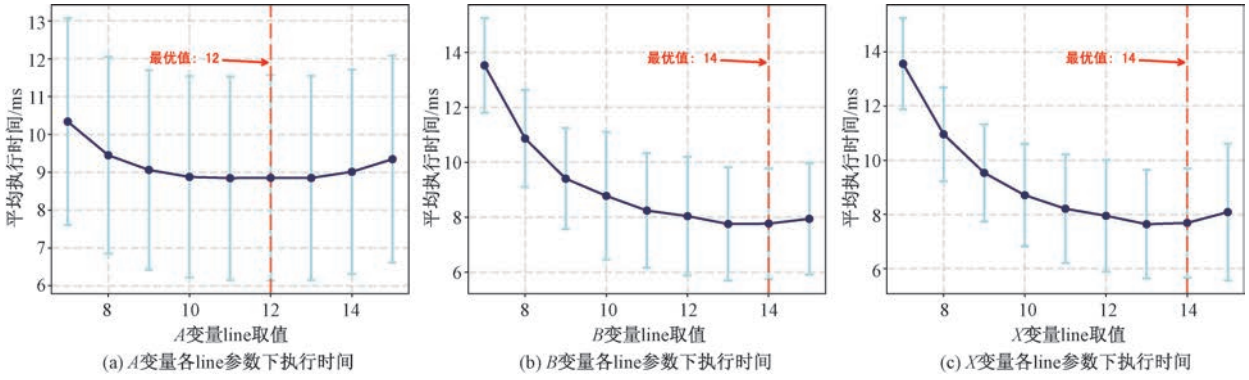

图 10 单缓冲区策略参数验证结果

图 10 展示了三个缓存参数分别对执行性能的影响，能够直观呈现每个参数在控制其他变量的情况下对性能的独立贡献。从图中可以观察到，B 变量和 X 变量的缓存参数对执行性能有显著影响，而 A 变量的影响相对较小，这是由于在 adi_kernel1 中，B 和 X 数组在计算过程中频繁被读写操作，而 A 数组主要用于只读访问。实验测得的最优配置为(12,14,14)，与理论模型预测的(14,14,13)不同，但性能差异极小，仅约 0.5%，验证了模型预测参数的准确性。

直接映射策略参数验证:选择 JACOBI2D 算子 jacobi_kernel1 核函数中的 A 变量,在 STANDARD 负载下进行验证。基于 LADWIM 推演,最优参数应为($set=5$,$line=10$)。实验构建了完整的参数搜索空间,$line$ 参数取值为 6 至 15,set 参数从 0 开始递增,在排除不满足硬件约束的组合后,最终测试了 55 种有效配置组合,实验结果如图 11 所示。

图 11 展示了 set 和 $line$ 两个缓存参数对核函数性能的影响规律。从图中可以观察到性能较优的参数配置大多满足 set 与 $line$ 之和为 14 或 15,这表明对于 A 变量而言,较大的缓存区域能带来显著的性能提升。此外,$line$ 参数并非单调递增关系,存在最优值,而 set 参数在接近最大可用值时往往能获得最佳性能。实验测得的最优配置为($set=5$,$line=10$),与理论模型预测的($set=2$,$line=12$)存在一定差异,但热力图显示这两个配置点之间的性能差异极小(约 0.4%),验证了理论模型的预测精度和实用性。

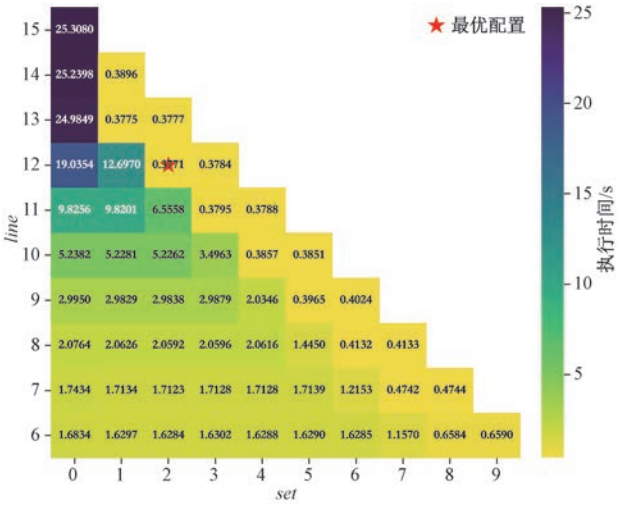

图 11　直接映射策略参数验证热力图

4.3　性能优化评估

4.3.1　基准测试集性能评估

本实验对 PolyBench 基准测试套件中的所有算子程序进行优化,综合评估访存优化框架的性能提升表现。各算子最大加速比结果如图 12 所示。

实验结果表明,缓存管理策略在大多数情况下能显著提升算子性能。在线性代数算法和数据挖掘相关算法中,如 MVT、GESUMMV 和 BICG 等算子,DSP 缓存优化版本表现最为突出,最高加速比可达 73 倍。这些核函数普遍都有极高的访存密度和极好的空间局部性,表明缓存管理策略在规则的数据访问模式和高并行度计算中具有显著优势。

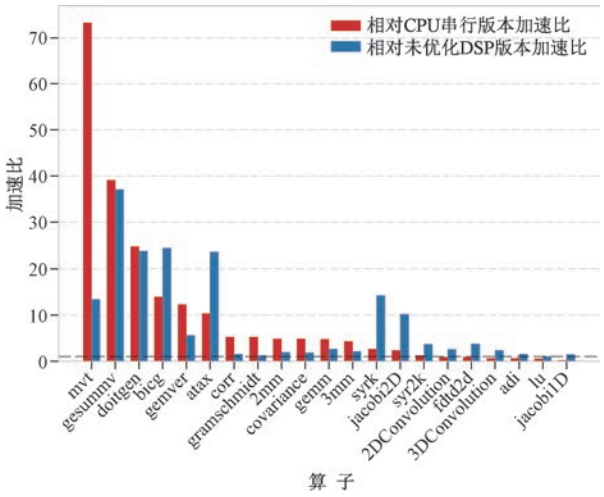

图 12　各算子最大加速比

在卷积算法中,缓存优化提供了稳定的 1.5 至 2.5 倍加速比。在模板计算算法中,JACOBI2D 算子能够达到最高 10.19 倍的缓存优化加速比。工作负载规模对性能的影响同样显著,在多数基准测试中,随着工作负载从 MINI 增长到 LARGE,DSP 缓存优化的加速比呈现上升趋势。

各算子程序在五个计算负载下相对于 DSP 原始版本的加速比结果如图 13 所示。从图中可以看出,对于绝大多数算子都表现出了很好的加速效果,最高加速比为 37.11 倍,最低也有 6%的性能提升。

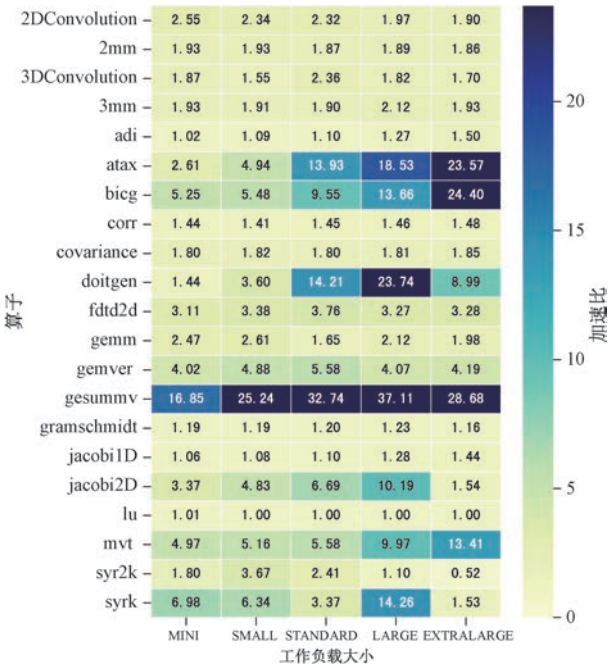

图 13　各算子在不同负载下相对 DSP 原始版加速比

4.3.2　部分弱优化效果算子原因分析

实验结果表明,虽然本文框架在大多数算子上取得了显著的性能提升,但部分算子的优化效果相

对有限。通过深入分析这些算子的访存特征、计算特性和硬件约束，识别出若干影响优化效果的关键因素。

(1)小数据规模下的缓存效率问题：在卷积算法2DCONV和3DCONV的小规模工作负载测试中，优化效果不如预期。这类算法的核函数中存在访存步长较大的变量，当数据规模较小时，可以缓存到SM中的数据量有限，每次缓存加载只能覆盖少量的数据行，数据在SM中的复用率低，缓存命中率不高。同时，小规模数据的计算量本身较少，缓存管理的额外开销(包括DMA通信、地址计算和替换逻辑)在总执行时间中占比较大，最终导致优化后的性能不佳。随着数据规模增大到STANDARD及以上负载，计算量大幅增加，缓存的有效性逐渐显现，优化效果达到1.5至2.5倍的稳定加速比。

(2)模板计算的迭代特性带来的启停开销：模板计算类算法，如ADI、FDTD-2D、JACOBI，普遍性能提升有限。这主要源于模板计算算法的迭代特性，即主机端需要多次调用核函数进行迭代计算，每次调用都涉及线程组的创建和销毁。MT-3000的线程启停开销累积在高频迭代调用中被放大，虽然单次迭代中访存优化能够带来一定的性能提升，但总体性能受限于迭代控制的开销。

(3)特殊访存模式下的策略失效：LU矩阵分解算法的加速比约为1，既无性能提升也未带来负优化。该算子的热点核函数lu_kernel2中仅有一个可优化变量，且该变量的访存模式极为随机，没有明显的步长规律，导致缓存预取失效、缓存命中率极低。LADWIM模型对该变量计算的空间局部性接近于0，但由于部分访存仍存在少量重复，模型未将其完全标记为“缓存策略不适用”。实际应用中，对该变量进行缓存管理既无法提升性能，也不会引入明显的额外开销，最终表现为加速比为1。此外，该算子的另一个特点是计算量随迭代递减，后期线程的计算负载严重不均衡，启停开销进一步掩盖了访存优化的效果。

(4)超大规模负载下的缓存压力：在EXTRALARGE工作负载测试中，部分算子，如DOITGEN和SYR2K，出现了性能异常现象，加速比相对LARGE负载未能进一步提升。这主要由于EXTRALARGE负载的参数呈指数增长，涉及的二维或三维数组规模庞大，批量缓存策略已无法适用，单缓冲区和直接映射策略虽然仍可使用，但缓存集合数量和缓存行大小的限制使得缓存覆盖率不足，最终导致优化效果下降。尽管如此，相对于未优化的核函数，EXTRALARGE负载下的优化版本仍能带来可观的性能提升，说明框架在极端场景下仍具有一定的适用性。

4.3.3 实际应用性能评估

本实验对航空发动机内流高效数值模拟平台应用X-Blue进行分析和优化，在单节点和多节点并行计算环境下评估访存优化框架的表现。

热点函数分析：通过单节点单进程执行10次迭代，统计各函数的执行时间及占比。结果显示，*gauss_all_seidel_gpu* 占总体执行时间的66.56%，是X-Blue应用的主要热点函数。在该函数调用的四个核函数中，*cal_new_f* 为热点核函数，时间占比超过80%。

访存特征分析与优化：使用MACE-C对三个核函数进行代码插桩并运行，获取每个变量的访存特征，再通过LADWIM模型进行推演。结果显示，除 *cal_new_f* 和 *init_b0* 的 f 变量外，其他变量均为步长为1的顺序访存，推演出的策略主要为单缓冲区策略。

单节点性能验证：按照推演结果对三个核函数进行优化，实验结果如表6所示。*init_b0* 和 *cal_new_f* 由于内部计算过程相似，均表现出接近3倍的加速比；*dx2f* 的加速比约为4倍。*gauss_all_seidel_gpu* 的执行时间从433.49s减少至148.83s，加速比达2.91倍，X-Blue的整体执行时间从198.23s减少至106.17s，性能提升87%。

表6 X-Blue应用核函数优化效果

核函数名	优化前时间(s)	优化后时间(s)	加速比
cal_new_f	359.73	128.82	2.79
init_b0	36.25	12.52	2.90
dx2f	5.58	1.41	3.96

多节点扩展性验证：在多节点环境下进行弱可扩展性测试，根据X-Blue采用的分层几何区域分解法，随着区域数量的增加，计算节点数量也同比例增加。实验结果如表7所示，当节点数为256时，共有16384个控制核，程序在万核规模下性能提升97%，所有配置下的加速比均稳定在1.95～2.03之间，表明框架在不同规模的分布式计算环境中均能有效提升性能。

表7 X-Blue应用多节点扩展性测试结果

区域数	节点数	优化前时间(s)	优化后时间(s)	加速比
4	32	874.56	431.12	2.03
8	64	870.18	446.67	1.95
16	128	889.69	443.52	2.01
32	256	877.17	445.19	1.97

4.4　与传统手工优化访存方法的对比分析

为了全面评估本文框架的性能优势,本节将框架与已有的手工缓存优化方法进行详细对比。对比方法采用张元胤等[10]的等分缓存管理策略,这是一种在 MT-3000 处理器上针对可视化算法提出的手工优化方法,代表了当前我国众核处理器手工优化访存性能的典型水平。

等分缓存管理策略的核心思想是将 SM 空间按照数据类型或访问维度进行等比例划分,为不同类型的数据分配独立的缓存区域。具体而言,该方法采用分块处理的方式,定义固定大小的批处理块,为每种数据类型,如系数矩阵 $\boldsymbol{ap}$、常数项 con、解向量 $\boldsymbol{f}$ 等,分配独立的 SM 缓存空间。在计算过程中,通过 scalar_load 和 scalar_store 指令显式地将数据批量加载到 SM 空间,完成计算后再批量写回主存。这种方法通过减少主存访问次数来提升性能,但需要开发者手动管理多个缓存区域的分配、加载和存储操作,编程复杂度较高。

本实验在 X-Blue 应用上应用等分缓存管理策略进行了优化,并在完全相同的硬件环境和测试数据集上进行。X-Blue 的网格规模为 1.8 亿,在这样规模和复杂度的实际应用软件上验证框架有效性更具实际意义。实验配置与前述实际应用测试保持一致,测试规模从 4 个节点扩展到 32 个节点。对比方法的实现严格遵循文献[10]中的优化策略,批处理块大小设置为 1024,SM 空间按照变量数量等分为多个独立缓存区。对比过程中,X-Blue 算法代码、网格规模、边界条件和求解器设置等条件完全相同,仅缓存管理策略不同以实现相同优化目标的实验设置对比。

图 14 展示了三种方法在不同规模下的性能对比,包括未优化的基准版本、等分缓存策略优化版本和本文框架优化版本。实验结果表明,等分缓存策略相对于基准版本实现了 15%～18%的性能提升,加速比在 1.15～1.18 之间。相比之下,本文框架的加速比达到 1.95～2.03,性能提升幅度为 95%～103%。与等分缓存策略相比,本文框架的加速比提升了约 68.5%～72.9%。

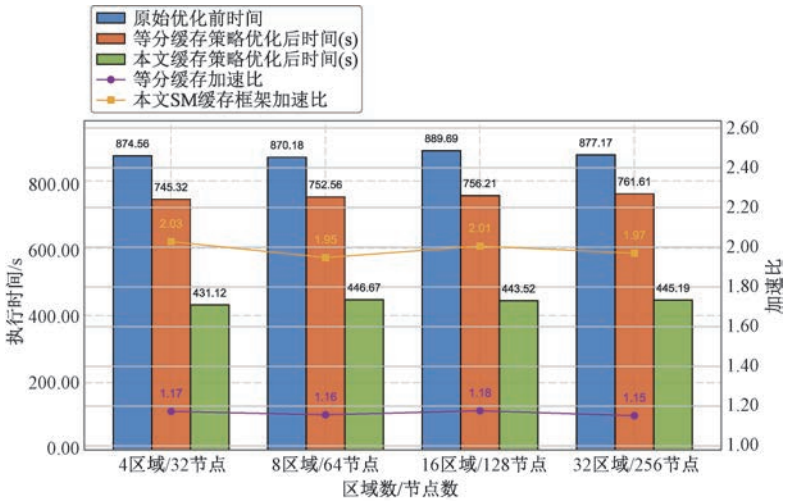

图 14　与传统手工优化方法的性能对比

与等分缓存策略相比,本文框架的性能优势与方法优越性体现在以下几个方面:

(1)缓存管理粒度:等分缓存策略采用固定大小(1024 元素)的批量加载方式,无论实际需要多少数据都会加载整个批次,导致 SM 空间利用率不高。当某次计算只需要访问少量数据时,仍然需要加载整个批次,造成带宽浪费。相比之下,本文框架根据变量的访存特征获取其访存范围,设置合理的缓存大小,实现按需的细粒度缓存管理,避免了不必要的数据搬运,SM 空间利用率更高。

(2)编程复杂度:等分缓存方法需要开发者显式管理多个缓存区域的分配、数据加载和存储操作,代码编写和调试难度较大,且容易出错。开发者需要仔细规划每个变量的缓存区大小,手动插入 scalar_load 和 scalar_store API 调用,还要处理批次边界情况,增加了代码的复杂性。本文框架通过 MT-Cache 封装底层缓存管理细节,通过 LADVIM 自动化地完成策略选择和参数配置,显著降低了优化门槛,提高了开发效率。

(3)自适应能力:等分缓存方法的参数,如批处理块大小、缓存区划分比例等,通常是根据经验或简单的试验确定的,缺乏系统化的方法指导,难以针对

不同的应用场景进行自适应调整。当应用的访存模式发生变化时,需要重新手工调整参数。本文框架通过 LADWIM 模型自动分析访存特征并推演最优策略和参数,能够根据不同应用的特点自适应地选择合适的优化方案,通用性和鲁棒性更强。

通过与手工优化方法的详细对比分析,验证了本文框架在性能、易用性和通用性方面的显著优势。实验结果表明,虽然手工优化方法能够获得一定的性能提升,但其效果有限且需要较高的优化成本。本文框架通过自动化的特征驱动优化,不仅实现了更高的性能提升,还大幅降低了优化门槛,为 MT-3000 平台的访存优化提供了更加高效和实用的解决方案。

4.5 框架开销分析

框架的工作流程包括四个步骤:MACE-C 对源代码进行插桩、运行插桩后的程序收集访存特征、分析输出文件生成访存特征向量、LADWIM 根据特征向量推演优化策略。本节对各步骤的时间开销进行量化分析。

插桩操作开销:MACE-C 基于 Clang AST 对源代码进行插桩,该过程在毫秒级完成。对于 X-Blue 应用和 PolyBench 中的典型核函数,插桩操作耗时均不超过 100 ms。

运行插桩代码开销:运行插桩后的程序是框架的主要时间开销来源。以 PolyBench 中 ADI 算子的 STANDARD 负载为例,未插桩的原始程序执行时间为 0.0875s,插桩后的程序执行时间为 54s,时间开销为原程序的 617 倍。这一显著开销源于该算子包含大量访存操作,每次访存都会触发特征计算和数据记录。对于 X-Blue 应用,单节点单次迭代的未插桩执行时间为 10.05s,插桩后执行时间为 44.59s,时间开销为原程序的 4.4 倍。

特征分析开销:分析访存特征输出文件并生成特征向量是另一主要开销。ADI 算子生成的访存特征原始记录文件大小为 20.3MB,分析该文件并生成访存特征的时间为 2.33s。X-Blue 应用生成的访存特征文件大小为 21.5MB,分析时间为 2.51s。该步骤的时间复杂度与访存记录数量呈线性关系。

策略推演开销:LADWIM 模型基于访存特征向量进行策略推演,该过程在毫秒级完成。对于 X-Blue 应用和 PolyBench 中的典型核函数,推演时间均不超过 50 ms。

存储开销:动态分析收集的访存特征数据以文本形式临时存储,文件大小通常在数十 MB 级别。这些数据文件仅在特征提取和分析阶段使用,分析完成后即可删除,对系统存储状态无持续影响。

框架开销对分析准确性的影响:尽管特征收集阶段引入了显著的时间和存储开销,但这些开销不影响访存特征分析结果的准确性,主要基于以下两方面的理论保证:

(1)插桩的语义透明性:MACE-C 的插桩代码仅在访存点插入监测逻辑,严格保持程序原有的控制流和数据流不变。插桩不修改循环边界、分支条件和数据依赖关系,仅在数组下标访问、指针解引用等访存表达式后追加记录函数调用。因此,插桩后程序执行的访存地址序列与原始程序完全一致,确保了特征采集的完整性和准确性。在前文基于 PolyBench、X-Blue 等的实验中,所有算子在主机端均设置了结果校验逻辑,插桩后的程序运行完成后均通过了正确性校验,验证了插桩未改变程序的计算语义。

(2)访存特征与执行时间的独立性:访存特征由程序的算法逻辑和输入数据共同决定,与程序的执行速度无关。以 ADI 算子为例,无论执行时间快慢,变量 A 从 $A[i][j]$到 $A[i][j+1]$的访问步长始终为 1,步长分布的统计结果不受监测开销影响。算法 2 通过实时计算相邻访存地址的差值来确定步长,这一计算过程独立于程序的整体执行效率,因此时间开销的增加不会影响最终提取的访存特征。

存储开销的优化分析:结合算法 2 的设计,当前方案已在算法层面采取了多项措施控制存储开销:采用固定大小的步长数组,最多记录 16 种访存模式,避免存储空间无限增长;通过实时更新计数器替代存储每次访存的原始地址,将单变量的空间复杂度控制在常数级别;设置步长上限阈值过滤异常跳跃访问,减少噪声数据干扰;在特征分析阶段采用频率阈值过滤低占比模式,确保输出聚焦于主要访存特征。

若实际应用中需进一步降低存储开销,可从以下两个方向进行优化:一是采用采样记录策略,以固定比例对访存操作进行采样,在大量访存操作的统计条件下,可在牺牲少量统计精度的前提下降低记录开销;二是采用二进制存储格式,将当前的文本格式输出替换为紧凑的二进制编码,可有效减少文件体积并提升读写效率。这些优化方向可根据具体应用场景的存储约束灵活实现,在保证分析准确性的前提下实现开销与精度的平衡。

开销合理性分析:尽管特征收集阶段引入了显

著的时间开销,但这一成本在实际应用场景中是完全可接受的,主要基于以下考虑:

(1)一次性投入:访存特征提取只需在优化前执行一次,后续优化代码的运行完全不包含任何监测开销,优化后的程序可长期高效运行。

(2)收益远大于成本:实验结果表明,优化后的程序可获得 2～73 倍的性能提升。以 X-Blue 为例,特征提取的总时间开销为 47.10s,而优化后每次迭代可节省约 92s,仅需运行不到 1 次迭代即可收回特征提取的时间成本。对于在实际生产环境中需要运行数千次甚至数万次迭代的应用,长期运行的优化收益相对于一次性的特征提取开销来说极为显著。

(3)可离线执行:特征提取和策略推演均可在开发和调试阶段完成,不影响生产环境的性能,这种离线分析模式使得框架的使用成本主要体现在开发阶段。

综上所述,框架引入的开销在实际应用中具有良好的性价比,特别是对于需要长期运行或多次执行的 HPC 应用,优化收益远超特征提取的一次性成本。

4.6　实验结果分析

实验结果表明,本文提出的特征驱动缓存加速访存优化框架在多个维度表现出良好的性能:

(1)策略选择准确性高:在所有测试场景中,LADWIM 模型推演的策略选择均与理论分析一致,验证了基于局部性感知和密度加权的策略推演方法的有效性。

(2)参数推演精度良好:模型推荐的参数配置与穷举搜索得到的最优参数非常接近,性能差异不超过 0.5%,表明数学建模方法能够准确捕捉访存行为与缓存性能之间的关系。

(3)性能提升效果显著:在 PolyBench 基准测试中,框架对线性代数和数据挖掘类算法最高可达 73 倍加速比;在 X-Blue 实际应用中,单节点实现了 87%的整体性能提升,在万核规模下性能提升 97%。

(4)通用性和可扩展性强:框架适用于不同类型的计算负载和不同规模的并行环境,为 MT-3000 平台的访存优化提供了系统化的解决方案。

5　结论与未来工作

本文针对 MT-3000 众核处理器访存优化面临的高门槛与低通用性挑战,提出了一种特征驱动缓存加速访存优化框架,系统性地解决了引言中提出的三个关键问题。首先,设计实现了基于 Clang AST 的 MACE-C 访存特征提取模块,采用动静结合的分析方法,实现了对程序中单个变量全生命周期访存行为的精确刻画,以解决访存特征表征粗粒度的问题。其次,构建了包含批量缓存、单缓冲区和直接映射三种互补策略的 MT-cache 自适应缓存管理策略库,通过统一接口设计有效提升了易用性和通用性。最后,提出了 LADWIM 策略与参数推演模型,通过访存特征向量构建和多变量联合优化算法,实现了从访存特征到缓存配置的全自动推演,摆脱了对人工经验的依赖。实验结果表明,该框架在 PolyBenchMT 基准测试集上对线性代数类算法最高可达 73 倍加速比,在航空发动机内流数值模拟应用 X-Blue 上实现了单节点 87%的整体性能提升,在万核规模下性能提升达 97%,与同类手工优化方法相比,本文框架的加速比提升最高约 72.9%,充分证明了特征驱动自动化优化方法的有效性和优越性。验证实验证实了策略选择的正确性和参数推演的准确性,性能差异不超过 0.5%。

本文进一步工作将基于策略选择和参数配置优化建议结果,研究结合大模型自动生成优化代码的方法,实现从源代码到优化代码的全自动优化。此外,当前框架主要针对控制核私有的 SM 进行优化,未来可以扩展到 AM、GSM、HBSM 等共享存储层次。GSM、HBSM 共享存储的优化需要设计相应的缓存一致性维护机制和多核协同访问策略,以应对更复杂的数据共享场景,这将是后续研究的重要方向。

致　谢　本论文由国家重点研发计划项目资助,项目编号为 2023YFB3001804。

参　考　文　献

[1] Domke J, Vatai E, Gerofi B, et al. At the locus of performance: Quantifying the effects of copious 3D-stacked cache on HPC workloads. arXiv, Technical Report: 2204.02235, 2023

[2] Lu K, Wang Y, Guo Y, et al. MT-3000: A heterogeneous multizone processor for HPC. CCF Transactions on High Performance Computing, 2022,4(2):150-164

[3] Chen J, Liu C, Luan Z, et al. Large-scale parallelization and optimization of lattice QCD on Tianhe new generation supercomputer//Proceedings of the 2023 IEEE International Conference on High Performance Computing and Communica-

tions, Data Science and Systems, Smart City and Dependability in Sensor, Cloud and Big Data Systems and Application. Melbourne, Australia, 2023:499-506

[4] Li J, Gu H, Zhao J, et al. Transplantation and optimization of molecular dynamics simulation on MT-3000. Future Generation Computer Systems, 2024,153:262-275

[5] Luo Y, Yang B, Liu J, et al. MT-office: Parallel password recovery program for office on domestic heterogeneous multicore processor. CCF Transactions on High Performance Computing, 2023,5(3):231-244

[6] Marinelli T, Gómez Pérez J I, Tenllado C, et al. COMPAD: A heterogeneous cache-scratchpad CPU architecture with data layout compaction for embedded loop-dominated applications. Journal of Systems Architecture, 2023,145:103022

[7] Park J, Jamil S, Khan A, et al. ScaleML: Machine learning based heap memory object scaling prediction//Proceedings of the 9th Non-Volatile Memory Systems and Applications Symposium. Seoul, Republic of Korea, 2020:1-6

[8] Fang J, Zhang P, Huang C, et al. Programming bare-metal accelerators with heterogeneous threading models: A case study of Matrix3000. Frontiers of Information Technology and Electronic Engineering, 2023,24(4):509-520

[9] Shi Y, Chen Z Y, Sun H Y, et al. Design of independent software stack of FT-Matrix DSP. Computer Engineering and Science, 2024,46(6):968-976 (in Chinese)
(时洋,陈照云,孙海燕等.面向飞腾迈创 DSP 的自主软件栈设计.计算机工程与科学,2024,46(6):968-976)

[10] Zhang Y Y, Xiao M G, Liu Z Y, et al. Optimization of isoline and isosurface extraction algorithm based on domestic heterogeneous many-core processors. Computer Engineering and Science, 2025,47(2):200-209 (in Chinese)
(张元胤,肖敏广,刘志勇等.基于国产异构众核处理器的等值线与等值面提取算法优化.计算机工程与科学,2025,47(2):200-209)

[11] Lu Y, Luan Z Z, Li G, et al. Multi-level parallel optimization of multi-head attention mechanism for the MT-3000 heterogeneous processor. Chinese Journal of Computers, 2025, 48(9):2049-2063 (in Chinese)
(路瑶,栾钟治,李根等.面向迈创 3000 异构处理器的多头注意力机制多重并行优化.计算机学报,2025,48(9):2049-2063)

[12] Wang H T, Sun Y F, Sui Y C, et al. MTTorch: PyTorch arithmetic library implementation and optimisation for the MT-3000 chip and Transformer models. Journal of Software, 2025,36(8):3896-3916 (in Chinese)
(王昊天,孙羽菲,隋轶丞等. MTTorch:面向 MT-3000 芯片和 Transformer 模型的 PyTorch 算子库实现与优化.软件学报,2025,36(8):3896-3916)

[13] Cheng D, Li G, Song A, et al. The implementation and optimization of FFT calculation based on the MT-3000 chip//Proceedings of the 3rd BenchCouncil International Symposium on Intelligent Computers, Algorithms, and Applications. Sanya, China, 2024:29-40

[14] Li R, Liu J. A heterogeneous KBA parallel algorithm for the Cartesian discrete ordinates for multizone heterogeneous system//Proceedings of the 8th International Conference on Computer and Communication Systems. Guangzhou, China, 2023:565-571

[15] Li R, Wang Q, Liu J. A heterogeneous parallel algorithm for the Cartesian discrete ordinates for multizone heterogeneous system. The Journal of Supercomputing, 2025,81(4):593

[16] Luo Y, Liu J, Xiao T, et al. Parallel implementation of SHA256 on multizone heterogeneous systems//Proceedings of the 2023 IEEE International Symposium on Parallel and Distributed Processing with Applications. Wuhan, China, 2023:416-422

[17] Luo Y, Liu J, Gong C, et al. An efficient heterogeneous parallel password recovery system on MT-3000. The Journal of Supercomputing, 2025,81(1):38

[18] Yin G, Tang Y, Xie W, et al. Applying binary code similarity detection on acceleration processor//Proceedings of the 4th International Conference on Computer Engineering and Intelligent Control. Zhuhai, China, 2023:454-459

[19] Fu H, Liao J, Yang J, et al. The Sunway TaihuLight supercomputer: System and applications. Science China Information Sciences, 2016,59(7):072001

[20] Wang C, Liu F F, Ma W J, et al. Research on a new matrix storage format and SpMV algorithm for the SW26010-Pro many-core processor. Chinese Journal of Computers, 2025, 48(6):1290-1304 (in Chinese)
(王萃,刘芳芳,马文静等.面向 SW26010-Pro 众核处理器的新型矩阵存储格式及稀疏矩阵向量乘(SpMV)算法研究.计算机学报,2025,48(6):1290-1304)

[21] Gschwind M. Chip multiprocessing and the Cell Broadband Engine//Proceedings of the 3rd Conference on Computing Frontiers. Ischia, Italy, 2006:1-8

[22] Zhao L, Yu Y B, Gao L M, et al. Application verification of parallel computing method for large-scale CFD numerical simulation of turbomachinery. Journal of Propulsion Technology, 2024,45(11):19-29 (in Chinese)
(赵磊,俞一波,高丽敏等.叶轮机械大规模 CFD 并行计算方法应用验证.推进技术,2024,45(11):19-29)

LI Huai-Jin, M. S. His main research interests include high-performance com puting and computer architecture.

DONG Xiao-She, Ph. D., professor, Ph. D. supervisor. His main research interests include high-performance computing, high-efficiency storage systems, and cloud computing.

CAO Jun-Kai, M. S. candidate, main research interests focus on high-performance computing.

DOU Shui-Tao, M. S. candidate. His main research interests focus on high-performance computing.

LIU Yu-Hang, M. S. candidate. His main research interests focus on high-performance computing.

WANG Qiang, Ph. D. candidate. His main research interests include high-performance computing and computer architecture.

BAI Xiu-Xiu, Ph. D., associate professor. Her main research interests include high-performance computing, computer vision, and artificial intelligence.

Background

Memory access optimization for many-core processors represents a critical challenge in the field of high-performance computing (HPC). As the performance gap between processor computational capabilities and memory access speeds continues to widen, the "memory wall" problem has become increasingly prominent, particularly in many-core environments where multiple computing cores compete for limited memory bandwidth simultaneously.

Internationally, memory access optimization research has primarily focused on GPU architectures and traditional multi-core processors, with optimization methods including specialized memory optimization, register blocking, memory coalescing, spatial blocking, and data prefetching techniques. However, these approaches are typically designed for specific hardware architectures and require deep understanding of computer architecture, making them difficult to directly apply to domestic many-core processors like MT-3000.

This paper addresses the high threshold and lack of universal methods for memory access optimization on MT-3000 many-core processors by proposing a feature-driven cache acceleration framework. The framework achieves up to 73× speedup for linear algebra algorithms on PolyBench benchmark suite and realizes 87% overall performance improvement on the X-Blue aero-engine numerical simulation application, with 97% performance enhancement at many-core scale, compared with similar traditional manual optimization methods, the proposed framework achieves a maximum speedup improvement of approximately 72.9%.

Within this broader research framework, our work specifically targets the memory access optimization challenges that significantly impact the performance of large-scale parallel applications on domestic many-core processors. This paper's contribution provides an automated feature-driven optimization approach that not only enhances the performance potential of MT-3000 processors but also establishes a foundation for developing comprehensive performance optimization toolchains for new-generation domestic supercomputing systems, ultimately supporting the strategic goal of technological independence in high-performance computing infrastructure.

Within this broader research framework, our work specifically targets the memory access optimization challenges that significantly impact the performance of large-scale parallel applications on domestic many-core processors. This paper's contribution provides an automated feature-driven optimization approach that not only enhances the performance potential of MT-3000 processors but also establishes a foundation for developing comprehensive performance optimization toolchains for new-generation domestic supercomputing systems, ultimately supporting the strategic goal of technological independence in high-performance computing infrastructure.