软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007488] [CSTR: 32375.14.jos.007488]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

SWTuner: 基于机器学习方法的分布式编译调优框架*

周文浩[1], 沈 莉[1], 王 飞[3], 肖 谦[1], 李 斌[2], 高秀武[2], 宋长明[1], 安 虹[1], 漆锋滨[2]

[1](中国科学技术大学, 安徽 合肥 230026)

[2](江南计算技术研究所, 江苏 无锡 214083)

[3](清华大学, 北京 100084)

通信作者: 漆锋滨, E-mail: qifb116@sina.com

摘 要: 随着编译技术的不断进步, 现代编译器支持了更为丰富的编程模型和复杂的编译优化, 使得手动调整编译选项以获得最佳性能变得非常困难. 尽管已有多种自动化的编译调优方法被提出, 但是面对庞大的搜索空间, 传统的启发式搜索算法很难避免陷入局部最优解. 同时, 现有调优方法主要针对单核或多核架构设计, 这限制了它们在大规模并行计算系统中的应用. 为了解决这些问题, 设计并实现基于机器学习方法的分布式编译调优框架 SWTuner, 通过引入 AUC-Bandit 分布式元搜索策略、机器学习模型指导的性能预测以及基于 SHAP 的编译选项分析及筛选等技术手段, 有效提升了编译调优过程中的资源利用率和搜索效率. 实验结果显示, SWTuner 在神威新一代超级计算机上对典型测试用例的调优中表现出色, 相较于其他调优方法, 其不仅缩短了搜索时间, 还能够显著降低搜索过程中的实际运行功耗. 在调优过程中, SWTuner 所使用的随机森林模型显示出了良好的泛化能力和预测准确性, 并且在保证调优效果的前提下有效降低了搜索空间的维度, 为高性能计算中的自动编译调优提供了一个高效且可靠的解决方案.

关键词: 自动化调优; AUC-Bandit 算法; 分布式元搜索; 随机森林模型; Shapley 值

中图法分类号: TP314

中文引用格式: 周文浩, 沈莉, 王飞, 肖谦, 李斌, 高秀武, 宋长明, 安虹, 漆锋滨. SWTuner: 基于机器学习方法的分布式编译调优框架. 软件学报. http://www.jos.org.cn/1000-9825/7488.htm

英文引用格式: Zhou WH, Shen L, Wang F, Xiao Q, Li B, Gao XW, Song CM, An H, Qi FB. SWTuner: Distributed Compilation Tuning Framework Based on Machine Learning Methods. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7488.htm

SWTuner: Distributed Compilation Tuning Framework Based on Machine Learning Methods

ZHOU Wen-Hao[1], SHEN Li[1], WANG Fei[3], XIAO Qian[1], LI Bin[2], GAO Xiu-Wu[2], SONG Chang-Ming[1], AN Hong[1], QI Feng-Bin[2]

[1](University of Science and Technology of China, Hefei 230026, China)

[2](Jiangnan Institute of Computing Technology, Wuxi 214083, China)

[3](Tsinghua University, Beijing 100084, China)

Abstract: With the continuous advancement of compilation technology, modern compilers support richer programming models and more complex compilation optimizations, which makes manually adjusting compilation options for optimal performance extremely challenging. Although various automated compilation tuning methods have been proposed, traditional heuristic search algorithms often struggle to avoid being trapped in local optima when confronted with vast search spaces. Moreover, most existing tuning methods target single-core or multi-core architectures, limiting their use in large-scale parallel computing systems. To address these issues, this study designs and implements a distributed compilation tuning framework, SWTuner, based on machine learning methodologies. By introducing AUC-Bandit-based

* 基金项目: 国家重点研发计划 (2023YFB3001500)
收稿时间: 2024-09-27; 修改时间: 2025-05-05; 采用时间: 2025-06-12; jos 在线出版时间: 2025-10-29

distributed meta-search strategies, machine learning model-guided performance prediction, and SHAP-based compilation option analysis and filtering, the resource utilization and search efficiency during the compilation tuning process are significantly improved. Experimental results show that SWTuner performs excellently in tuning typical test cases on the new-generation Sunway supercomputer, not only reducing search time but also achieving notable reductions in actual execution power consumption during the search process compared to other tuning methods. During the tuning process, the random forest model employed by SWTuner demonstrates good generalization capability and prediction accuracy, effectively reducing search space dimensionality while maintaining tuning effectiveness, providing an efficient and reliable solution for automatic compilation tuning in high-performance computing.

Key words: auto-tuning; AUC-Bandit algorithm; distributed meta-search; random forest model; Shapley value

编译器作为底层硬件与上层应用之间的重要桥梁, 其实施的优化技术对于提升程序性能至关重要[1]. 例如, 循环展开[2]、常数传播[3]等优化通过减少不必要的计算来提升程序运行速度. 结构体重组[4]及函数内联[5]等优化通过调整数据和指令的空间排布来增强 Cache 局部性. 同时, 现代编译器还能够利用单指令流多数据流 (single instruction multiple data, SIMD) 指令集和自动并行化等技术充分挖掘特定硬件架构的并行处理能力[6−8].

由于编译器内部的优化逻辑极为复杂, 即便是经验丰富的程序员也很难全面掌握所有编译选项的具体作用及其相互影响. 手动调整编译选项以达到最佳性能通常是一项既耗时又费力的工作. 程序员需要深入了解每个选项的意义, 测试不同配置组合下的程序表现, 并反复试验以找到最优配置. 这一过程不仅需要大量的时间和精力, 而且结果往往受到个人经验和知识水平的限制, 难以确保获得最佳性能[9].

随着编译技术的发展, 编译器版本及语言标准快速迭代, 编程模型和抽象层次也越发复杂. 如表 1 所示, DPC++ 支持统一的 SYCL 异构编程模型[10], RUSTC 通过静态类型检查和所有权模型确保内存安全[11], 而 Julia 则通过内置的函数式编程以及多重派发等功能, 在保证代码编写简单、轻松易读的同时还能拥有与静态语言类似的性能[12]. 这些新特性带来了更为复杂的编译挑战, 需要更多的编译选项来支持. 然而, 选项之间复杂的相互作用以及对应用特点和硬件架构的依赖性, 使得传统的手工调优方法难以应对.

表 1　主流编译器支持的编程模型、编程语言及选项数量

编译器	版本	编程模型	编程语言	选项数量
GCC	11.4.0	OpenMP/OpenACC/OpenCL	C/C++/FORTRAN	252
LLVM	14.0.0	OpenMP/OpenACC/OpenCL	C/C++	391
DPC++	14.0.0	SYCL	C++	441
NVCC	11.5	CUDA	C/C++	123
RUSTC	1.75.0	—	RUST	47
GO	1.18.1	—	GO	66
Julia	1.8.1	—	Julia	37

为了解决上述问题, 研究人员设计了一系列自动化的编译调优方法, 以确保程序在特定的目标机器和体系结构上达到预期的性能标准[13]. 迭代式自动调优[14]是目前该领域最常用的技术之一, 其定义搜索空间并选择一组初始配置来启动迭代过程, 随后根据程序性能的变化不断调整选项及参数, 逐步将结果收敛到最优配置. 在搜索策略的选择上, 当前主流的编译调优方法通常依赖基于学习策略的变体, 包括遗传算法、模拟退火算法、粒子群算法或贝叶斯优化等启发式搜索算法, 以及基于机器学习或深度学习方法构建的性能模型等[15−17]. 这些自动化的调优方法不仅减轻了程序员的负担, 还能够更加高效地挖掘程序的潜在性能.

然而, 当前编译调优的研究仍面临以下几个方面的挑战: 首先, 目前主流的编译调优框架大多面向单核或多核系统设计, 这导致搜索过程难以有效利用大规模并行计算系统的硬件资源, 扩展性受到严重限制. 其次, 传统的搜索策略如遗传算法、模拟退火等容易陷入局部最优解, 而更高级的搜索策略往往依赖于使用基于机器学习方法构建的复杂而强大的模型. 这些模型 (例如深度神经网络等) 需要大量样本进行训练, 以保证模型的准确性和泛化性. 同时, 当前的搜索方法侧重于提高搜索效率和精度, 而忽视了搜索过程中的功耗开销. 在高性能计算领域, 功耗管理变得尤为重要, 因为这直接关系到系统的总体运行成本和可扩展性. 最后, 当前的搜索空间优化方法主要依赖于传统的统计学方法, 难以充分挖掘编译选项之间的深层次依赖关系, 从而导致错过潜在的优化机会.

为了应对上述挑战, 本文设计并实现了基于机器学习方法的分布式编译调优框架 SWTuner, 以提升调优过程的资源利用率和搜索效率. 本文的主要贡献如下.

1) 在国产超算系统软硬件环境的基础上, 实现了基于 AUC-Bandit 的分布式元搜索策略. 该策略通过分布式优化显著降低了元搜索过程中的编译和运行开销, 有效提升了元搜索技术的整体效率.

2) 使用随机森林模型对编译选项与程序性能的关系进行建模, 并将其应用于后续的元搜索过程. 该方案有效减少了搜索过程中程序实际执行的次数及时间, 从而降低了编译调优过程的能耗.

3) 引入 Shapley 值及 Shapley 交互值来评估编译选项的重要程度及其相关性. 通过逐步淘汰影响较小的编译选项, 在保证调优效果的前提下降低了搜索空间的维度, 进一步提升了搜索效率.

本文第 1 节介绍背景及相关工作. 第 2 节介绍 SWTuner 编译调优框架的基本结构及关键技术实现. 第 3 节通过实验对关键技术的有效性以及整体调优效果进行评测. 第 4 节给出总结和展望.

1 相关工作

目前, 编译器自动调优的研究主要集中在优化搜索空间、调整评估模型以及改进搜索策略这 3 个方面.

首先, 较大的搜索空间意味着需要考虑更多的编译选项组合, 这不仅增加了计算资源的需求, 还可能导致调优过程耗时较长. 为了提高编译调优的搜索效率, 研究者们提出了多种方法来优化搜索空间. 例如, MiCOMP 方法[18]将所有选项分为几个大组, 固定组内选项的出现次数和顺序, 仅调整大组之间的顺序和出现次数. 这种方法显著减少了搜索空间, 调优后的程序性能最多可提升 1.51 倍. ICMC 方法[19]通过构建选项关系依赖图来识别选项间的依赖关系, 并仅以强依赖的子序列作为调优单位, 从而显著缩小搜索空间的维度. Theodoridis 等人[20]为减少搜索空间, 将函数调用图划分成更小的子图来探索子图的最优内联参数, 这种方法使得穷举搜索成为可能.

其次, 有效的评估模型能够准确地预测不同编译选项组合对程序性能的影响, 这对于快速定位最优配置至关重要. MLGOPerf[21]是一种基于强化学习的迭代优化算法, 它通过 IR2Perf 模型从中间表示中提取特征来预测代码性能. AdaTune[22]使用自适应的评估方法, 通过监测程序执行的稳定性来决定何时停止测量, 以平衡准确性和效率. 部分自动调优方案结合使用成本模型和实际测量, 以提高效率和准确率. 例如, Ansor[23]先使用成本模型快速估计配置性能, 然后实际测量预测性能较好的配置, 并将测量结果用于进一步训练成本模型, 提高预测准确性. 通过不断迭代, 成本模型的预测准确率逐渐提高, 从而提升调优算法的整体效率.

最后, 搜索策略决定了每轮迭代后如何根据当前配置的性能选择下一个配置. 如果过于依赖某一种优化策略, 可能会导致陷入局部最优值, 错过新的提升程序性能的机会. SRTuner 方法[24]将搜索策略抽象为多臂老虎机模型, 其中每个选项代表一条“手臂”, 在每轮迭代过程中根据其性能调整对该分支的奖励. BOCA 方法[25]使用基尼系数评估选项的重要性, 并优先考虑重要选项. 同时, 为了避免过拟合, 该方法在每次迭代过程中还会随机选择一些不重要的优化选项, 与重要选项一起组成候选集合进行调优. Bliss 方法[26]使用多种具有不同代理模型和采集函数的贝叶斯优化模型, 迭代的过程中按照一定概率选择贝叶斯模型, 并在每轮迭代结束后为效果更好的模型分配更高的概率值.

OpenTuner[27]是一个用于构建特定领域多目标程序自动调优器的框架, 其核心优势在于灵活性和可扩展性. 与其他搜索框架相比, OpenTuner 能够更加方便地集成新的搜索算法, 从而增强其调优能力. 通过元搜索策略, OpenTuner 同时使用多种不同的搜索技术, 并根据每种技术的实际表现动态调整资源分配, 确保资源的有效利用, 提高整体调优过程的效率. 这一机制使得 OpenTuner 能够在有限时间内找到更优的编译选项组合. 同时, OpenTuner 还能应对多目标优化挑战, 如同时优化程序的执行时间和内存消耗. 因此, OpenTuner 成为当前编译调优领域的一个主流选择, 为研究人员提供了一个强大而灵活的工具.

然而, 现有的编译调优方法在仍面临着一些挑战. 首先, 在搜索空间方面, 现有方法往往难以在短时间内全面覆盖对程序性能影响显著的编译选项组合, 特别是对于那些包含复杂相互作用和非线性效应的选项. 其次, 在评估模型方面, 虽然已经提出了多种预测模型, 但它们的准确度仍受限于训练数据的数量和质量, 尤其是在处理新类型

的应用或架构时, 模型泛化能力不足的问题尤为突出. 此外, 现有的搜索策略虽然能够在一定程度上平衡探索与利用, 但在面对高度动态且不确定的编译调优任务时, 仍然容易陷入局部最优或收敛速度缓慢的问题.

2 SWTuner 编译调优框架的设计与实现

为了应对上述挑战, 本文基于 OpenTuner 设计并实现了编译调优框架 SWTuner, 旨在结合分布式优化策略和先进的机器学习技术克服现有方法的局限性. SWTuner 的基本结构如图 1 所示, 主要包括 AUC-Bandit 分布式元搜索模块、机器学习模型指导的性能预测模块以及基于 SHAP (Shapley additive explanations) 的选项分析及筛选模块等 3 个核心模块.

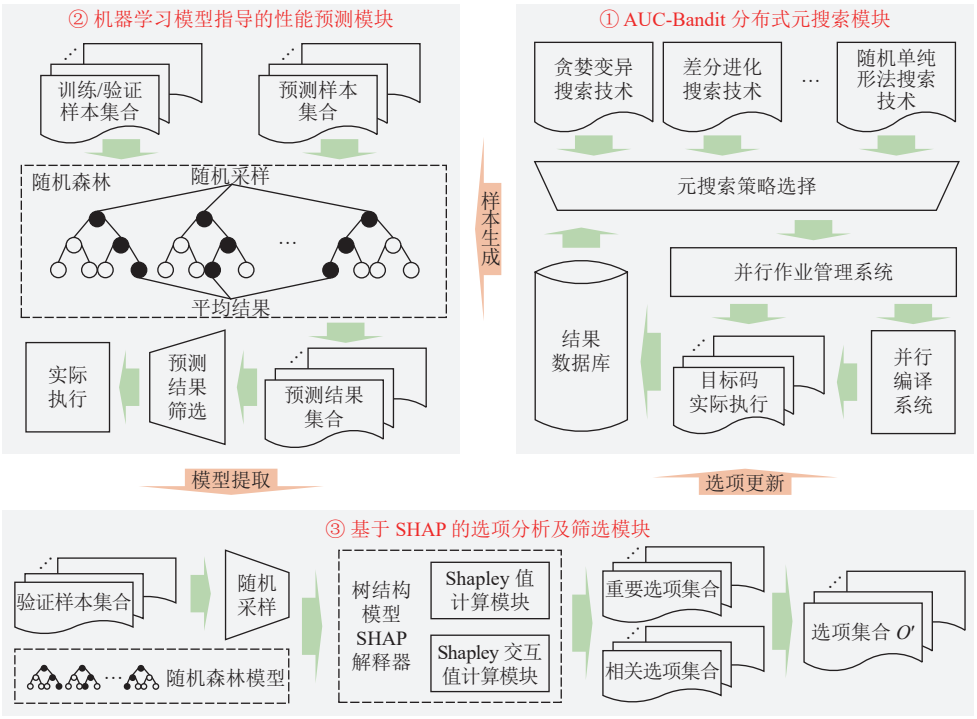


图 1 SWTuner 编译调优框架组成结构框图

AUC-Bandit 分布式元搜索模块作为 SWTuner 框架的核心, 以曲线下面积 (area under the curve, AUC) 作为评价指标, 动态地为多种启发式搜索技术分配信用. 在并行作业管理系统的统一管理和调度下, 编译系统根据给定的搜索样本集合对目标程序进行本地编译和链接, 生成一系列可执行文件, 并且在超级计算机系统上并行执行. 计算节点收集到的性能数据被存储至结果数据库, 以实现不同搜索技术之间的信息交互, 从而帮助提升元搜索策略的整体优化效果.

机器学习模型指导的性能预测模块致力于构建一个通过编译选项组合预测程序性能的模型. 为实现此目标, 该模块从结果数据库中提取训练和验证样本集合, 并采用随机森林模型作为基础机器学习模型. 模型训练完成后, 可以对新的样本进行预测, 从而缩短选项搜索过程中的编译及运行开销并降低能耗. 同时, 为避免由于预测误差导致错过最优解, 该模块通过执行阈值对性能预测结果进行筛选, 并且对具有优化潜力的选项组合进行实际执行验证.

基于 SHAP 的选项分析及筛选模块专注于评估各个编译选项的重要程度及其相关性. 该模块从结果数据库中抽取样本集合, 并基于已训练完成的随机森林模型构建一个针对树结构模型的 SHAP 解释器. 随后, 该模块计算样本集合的平均 Shapley 值及平均 Shapley 交互值, 并据此筛选出对程序性能影响显著的选项集合, 以指导下一轮迭

代的元搜索空间构建.

上述 3 个模块相互配合, 形成了一个完整且高效的自动化编译调优流程. 下面, 本文分别针对每个模块中采用的关键技术进行详细介绍.

2.1 基于 AUC-Bandit 的分布式元搜索

为了动态地分配计算资源给不同的搜索技术, 同时平衡探索与利用以避免陷入局部最优, SWTuner 将搜索过程抽象为赌博机 (bandit) 问题, 并采用曲线下面积指导具体搜索技术的选择. AUC-Bandit 元搜索策略的核心在于使用 AUC 值来评估每个搜索技术的表现, 并基于评估结果决定哪些技术应该获得更多的测试机会[27]. 具体而言, AUC_t 值反映了搜索技术 t 在滑动窗口中产生新的全局最佳结果的能力, 其计算方式如下所示:

$$AUC_t = \frac{2\sum_{i=1}^{|V_t|} iV_t[i]}{|V_t|(|V_t|+1)} \tag{1}$$

其中, V_t 是一个 Bool 类型的数组, 其元素 $V_t[i]$ 用于记录滑动窗口中第 i 次使用搜索技术 t 时是否获得全局最优结果 (是则为 1, 否则为 0). 滑动窗口通过限制历史数据的观察范围来动态调整搜索技术的资源分配, 窗口长度在 SWTuner 中通过参数 window 进行配置. $\sum_{i=1}^{|V_t|} iV_t[i]$ 表示 V_t 中各元素的加权和, 其权重是该技术在滑动窗口中的位置 (从 1 开始计数).

为了更好地理解 AUC 的含义, 本文将其直观地表示为图 2 所示的笛卡尔坐标系中的红色线条. 其中, 横坐标表示技术 t 在滑动窗口中的使用次数, 纵坐标表示 V_t 中每个元素的加权值, 而 $\sum_{i=1}^{|V_t|} iV_t[i]$ 则等价于曲线与横轴所包围区域的面积. 如果技术 t 在第 i 次使用中产生了新的全局最佳结果, 则曲线在 i 处向上跳跃, 否则保持不变. 通过将 AUC 值归一化到 [0, 1] 之间, 元搜索策略能够直接比较不同搜索技术之间的优劣程度.

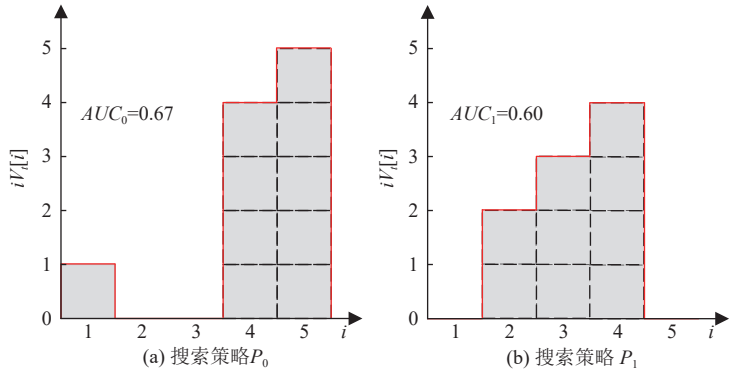


图 2 曲线下面积的图形化表示

在 SWTuner 元搜索驱动的初始化阶段, 所有搜索技术均被赋予相同的信用值. 而随着搜索过程的推进, 搜索技术被选择的概率将与它的得分成正比. 技术 t 的得分 $Score_t$ 的计算方式如下所示:

$$Score_t = AUC_t + EXP_t \tag{2}$$

公式 (2) 中除了 AUC_t 外, 还包括一个探索项:

$$EXP_t = C\sqrt{\frac{2\ln H}{|V_t|}} \tag{3}$$

其中, C 是控制探索与平衡的常数, H 是滑动窗口的长度. 探索项 EXP_t 的存在是为了确保即使某种搜索技术当前表现不佳, 也会有一定的概率被再次尝试, 以避免陷入局部最优.

SWTuner 的 AUC-Bandit 元搜索驱动支持多种不同的搜索方法, 包括均匀贪婪变异 (uniform greedy mutation)、正态贪婪变异 (normal greedy mutation)、差分进化变异 (differential evolution mutation) 以及随机单纯形法 (random Nelder-Mead) 等. 其中, 均匀贪婪变异通过在参数空间中均匀选择变异方向, 结合贪婪策略实现探索与利用之间的

平衡; 正态贪婪变异利用正态分布来指导参数变异; 差分进化变异是一种基于群体智能的技术, 通过个体间的差异引导搜索方向; 而随机单纯形法则是一种无需依赖梯度的过程, 通过调整多维三角形的顶点来迭代寻优. 这些多样化的搜索方法为解决复杂的编译调优问题提供了灵活的选择.

基于 AUC-Bandit 的元搜索策略通过综合考虑历史性能和加速效果, 能够在有限的时间内生成大量有潜力的搜索样本. 这种方法非常适合于超级计算机环境下的编译调优: 元搜索策略生成的大量搜索样本可以被同时分配到多个计算节点上进行编译和执行, 从而充分利用分布式计算资源提升搜索效率.

图 3 展示了 SWTuner 的分布式元搜索策略的执行流程. 首先, AUC-Bandit 元搜索驱动根据滑动窗口中各搜索技术的 *AUC* 值, 选择一组当前最优的搜索技术, 生成搜索样本集合$\{(x_i, t_i)$: i=1, 2,…, $p\}$, 其中 x_i 表示编译选项的组合, t_i 表示该选项组合的实测性能 (初始化为 *inf*), p 表示并行规模. 然后, 元搜索驱动将每个搜索样本 (x_i, t_i) 视作一个计算任务, 通过并行作业管理系统将任务提交到各个计算节点, 并根据可用资源和任务负载情况进行动态调度. 计算节点从全局文件系统中获取需要编译的源代码后, 按照给定的编译选项组合 $x_i=(o_1^i, o_2^i,…, o_n^i)$ 进行本地编译及运行, 完成对性能结果 t_i 的更新. 最后, 当计算节点完成任务后, 元搜索驱动会将搜索样本 (x_i, t_i) 上传至结果数据库. 在下一轮迭代中, 元搜索驱动从结果数据库中读取历史样本, 评估编译和运行的效果, 并根据评估结果动态地调整搜索技术, 从而形成了一个正向的反馈循环. 这种分布式的编译和运行机制充分利用了超级计算机的软硬件资源, 从而提高了元搜索策略的效率和质量.

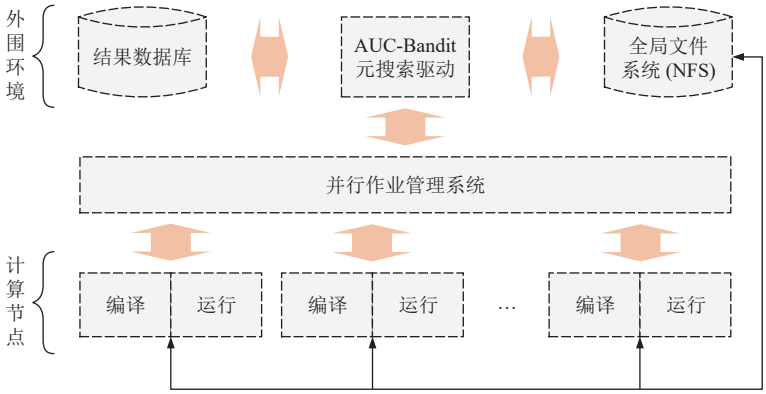


图 3 分布式元搜索策略的执行流程图

2.2 机器学习模型指导的性能预测

分布式优化能够有效缓解元搜索策略的性能瓶颈, 但是频繁地本地编译和运行也会引入额外的功耗开销. 随着超级计算机系统规模的不断增加, 功耗问题已经成为限制其可扩展性的主要瓶颈之一[28]. 本文通过对编译选项与程序性能的关系进行建模, 能够有效指导后续的调优过程, 从而降低实际执行所需的功耗.

目前针对性能评估的研究主要依赖于使用深度学习方法构建的复杂而强大的模型[15−17], 这些模型为了保证准确性和泛化性, 往往需要大量样本进行训练. 在真实的应用场景中, 被调优程序虽然可能会执行多次, 但是其输入数据的类型和大小往往会随着时间的推移而改变, 并且算法实现也可能发生增量改进, 从而导致最优编译选项组合的改变. 针对程序的某一版本或配置进行大量的真实采样, 需要占用大量的计算资源和机时, 这在实际的编译调优任务中往往难以满足[29].

随机森林是一种集成学习方法, 它通过构建多棵决策树来提高模型整体的预测精度[30]. 随机森林模型中的每棵决策树都是独立训练的, 并且在训练过程中使用了特定的技术来增加模型的多样性并减少过拟合风险. 相较于深度神经网络, 其资源占用更少, 训练和推理速度更快, 并且在稳定性以及小数据集适应性等方面均具有一定优势. 同时, 相比连续特征, 编译选项这样的二值或多值特征在决策树构建阶段的特征选择和数据分割中开销更小, 因此模型的训练效率更高.

本文使用随机森林模型对被调优程序的编译选项和性能之间的关系进行建模, 以指导该程序的后续调优. 为了直观地展示模型的构建过程, 本文以包含 5 个选项的编译调优任务作为示例. 假设每个编译选项 o_i 仅支持打开和关闭两种状态 (分别表示为 1 和 0), 且编译选项 o_i 与程序执行时间 *perf* 的关系满足以下约束:

$$perf = 20.0 - 5.0 \times o_1 - 4.0 \times o_2 + 3.0 \times o_3 - o_1 \times o_4 + 0.5 \times o_5 \tag{4}$$

公式 (4) 是一个假设的简化示例. 其中, 选项 o_1 和 o_2 为性能敏感选项, 选项 o_3 引入了额外优化开销, 选项 o_1 与 o_4 同时开启将削弱优化效果, 而选项 o_5 对性能影响较小. 此示例用于说明随机森林模型如何捕捉选项间的线性与非线性关系, 实际应用中模型能够通过历史样本自动学习此类模式. 首先, SWTuner 根据该约束进行随机采样, 获得如表 2 所示训练数据. 然后, 从训练数据中随机抽取部分样本, 根据基尼指数等指标选择最佳特征, 并根据该特征的不同取值将样本划分为若干子集. 对于每个子集, 重复上述特征选择和数据划分的过程, 就能够构造出一棵用于预测程序性能的决策树. 对于上述示例, 本文使用 3 棵决策树的平均预测结果作为最终的预测结果. 如图 4 所示, 该示例的随机森林模型预测精度较高, 相对误差仅为 1.8% 左右. 除了上述示例中二值选项, SWTuner 还支持循环展开次数、循环分块大小等多值选项. 一方面, SWTuner 继承了 OpenTuner 的 IntegerParameter 类, 能够根据给定的取值范围自动进行搜索采样. 另一方面, 随机森林模型天然支持对多值特征的建模, 决策树能够通过划分节点捕捉多值选项与性能的关系.

表 2　包含 5 个选项的编译调优任务的随机森林模型训练数据

编号	o_1	o_2	o_3	o_4	o_5	*perf*	编号	o_1	o_2	o_3	o_4	o_5	*perf*
1	0	0	0	0	0	20.0	9	1	0	0	0	0	15.0
2	0	0	0	0	1	20.5	10	1	0	0	0	1	15.5
3	0	0	0	1	0	20.0	11	1	0	0	1	0	14.0
4	0	0	1	0	0	23.0	12	1	0	1	0	0	18.0
5	0	1	0	0	0	16.0	13	1	0	1	0	1	18.5
6	0	1	0	0	1	16.5	14	1	1	0	0	0	11.0
7	0	1	0	1	0	16.0	15	1	1	0	0	1	11.5
8	0	1	0	1	1	16.5	16	1	1	0	1	0	10.0

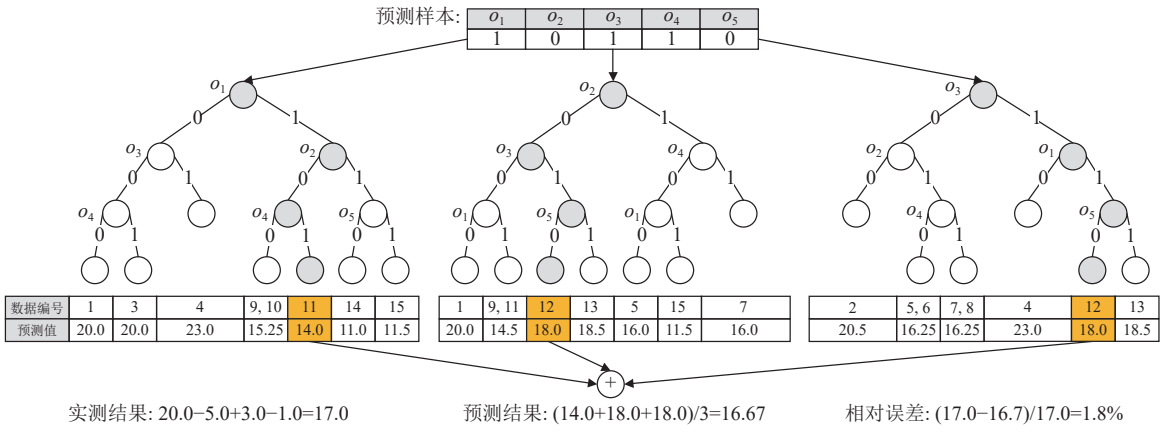


图 4　包含 5 个选项的编译调优任务的随机森林模型示例

通过复用结果数据库中已存储的历史搜索样本集合 $S_t=\{(x_i, t_i): i=1, 2,\ldots, W_t\}$, 能够非常方便地对随机森林模型进行训练. 同时, 为了评估模型的泛化效果, SWTuner 从结果数据库中随机抽取 W_v 个样本构成验证样本集合 S_v, 其中 $S_v \cap S_t = \varnothing$, 并使用决定系数[31]来度量 S_v 上模型预测结果的准确性. 决定系数 R^2 的计算公式如下:

$$R^2 = 1 - \frac{\sum_{i=1}^{W_v} (t_i - \hat{t}_i)^2}{\sum_{i=1}^{W_v} (t_i - \bar{t})^2} \tag{5}$$

其中, $\sum_{i=1}^{W_v}(t_i-\hat{t}_i)^2$ 为所有样本实测值与预测值之间的残差平方和 (residual sum of squares, RSS), $\sum_{i=1}^{W_v}(t_i-\bar{t})^2$ 为所有样本的总离差平方和 (total sum of squares, TSS). 当 R^2 的值接近 1 时, 表示预测值与实测值非常接近, 模型拟合效果很好. 当 R^2 的值接近 0 时, 表示模型的解释能力较弱, 无法准确表示编译选项组合与程序性能之间的复杂非线性关系.

当模型的预测精度达到一定水平后, 就可以在后续的搜索过程中对程序性能进行预测, 从而替代实际执行以降低功耗开销. 由于模型的泛化程度有限, 而程序的执行也存在一定的动态性, 因此预测的结果可能存在误差, 从而导致错过最优解的情况出现. SWTuner 通过执行阈值 (execution threshold, *ET*) 对模型的预测结果进行筛选, 并且对具有优化潜力的选项组合进行实际执行验证. 执行阈值的定义如下所示:

$$ET=\frac{\sum_{(x_i,t_i)\in S_t\cup S_v}t_i}{(W_t+W_v)\cdot\max(R^2,0.1)} \tag{6}$$

对于编译调优任务, 假设每个编译选项对程序性能的影响均为独立随机事件, 并且使用完全随机的搜索策略, 根据中心极限定理[32], 性能模型的预测结果应符合正态分布. 由于正态分布属于对称分布, 其概率密度曲线下均值左右两侧的面积相等, 假设程序在单位执行时间内的功耗为固定值, 则在理想情况下, 使用 *ET* 筛选后的功耗节省比例约为 50%. 考虑到模型预测结果的可信度与训练过程的质量密切相关, 本文将 R^2 引入 *ET* 的计算过程. 在一些极端情况下, 模型过拟合或者数据中存在异常值可能导致 R^2 的结果为负值, 因此公式 (6) 使用 $\max(R^2, 0.1)$ 作为分母. 当模型的预测结果较为准确时, *ET* 值接近历史实测结果的平均值, 而当模型的解释能力减弱时, R^2 值减小, 从而导致预测执行逐步退化为实际执行.

2.3 基于 SHAP 的选项分析及筛选

编程语言和编译技术的不断发展推动了编译器种类及编译选项数量的迅速增长. 可配置编译选项数量的增多导致所有可能的选项组合数量以指数形式增长. 因此, 尝试通过遍历整个搜索空间来寻找最优配置的方法, 在实际应用中往往是不可行的. 目前主流的编译调优框架通常采用启发式或近似算法来解决这一问题, 以便在合理的时间内找到接近最优解的解决方案. 然而, 这些方法仍然需要面对庞大的搜索空间, 在优化质量和搜索效率之间做出权衡. 如何在保证优化效果的前提下探索更加高效和适应性强的搜索空间优化策略, 是当前编译调优领域的重要研究方向.

不同的编译选项对程序性能的影响存在显著差异. 例如, 某些选项能够大幅提升程序的运行速度, 但可能会增加二进制文件的大小; 而其他选项可能减少内存使用量, 但以牺牲执行效率为代价; 还有一些编译选项则主要用于调试, 对程序性能几乎不产生直接影响. 上述差异性提示我们在选择编译选项进行优化时, 必须仔细考量应用程序的具体需求以及目标硬件的特性, 并将重点放在那些对调优目标有显著影响的选项上.

Shapley 值[33]是合作博弈论中的重要概念, 用于公平地计算每个玩家对联盟总收益的贡献. 在可解释机器学习领域, 通过计算 Shapley 值, 可以了解每个特征对模型预测值的贡献程度, 从而帮助理解模型的行为[34]. SWTuner 使用 Shapley 值来评估每个编译选项对程序性能的影响程度. 具体地, 对于编译选项组合 $x=(o_1, o_2,\ldots, o_n)$, 编译选项 o_i 的 Shapley 值的计算方式如下所示:

$$\phi_i=\sum_{S\subseteq O\backslash\{o_i\}}\frac{|S|!(n-|S|-1)!}{n!}(f_x(S\cup\{o_i\})-f_x(S)) \tag{7}$$

其中, $O=\{o_1, o_2,\ldots, o_n\}$是 x 中包含的所有编译选项的集合, n 为选项的个数, $S\subseteq O\backslash\{o_i\}$为不包含 o_i 的选项集合的子集, $f_x(S)$ 为随机森林模型对选项子集 S 的预测. $|S|!(n-|S|-1)!/n!$为子集 S 的权重, 其中分母 $n!$表示 n 个选项在任意排序情况下的组合数量, 分子$|S|!(n-|S|-1)!$表示在确定子集 S 后, n 个选项在特定排序的情况下的组合数量.

尽管 Shapley 值具有令人信服的理论优势, 但是其精确计算过程具有指数复杂度. SHAP 框架[35]针对树形模型推导了一种优化算法, 将计算精确 Shapley 值的复杂度从 $O(TL2^M)$ 降低至 $O(TLD^2)$. 其中, T 是决策树的数量, L 是所有树中的最大节点数量, M 是特征数量, D 是所有树的最大深度. 这种复杂度的指数级降低使得应用 Shapley 值评估编译选项的重要程度成为可能.

在编译优化过程中, 多个编译选项可能同时作用于一个优化遍. 例如, 对于第 2.2 节中包含 5 个选项的编译调

优任务示例, 同时启用 o_1 和 o_4 选项优化程序后的性能均优于单独使用 o_1 或 o_4 的性能, 即说明 o_1 和 o_4 选项存在相关性. 在这种情况下, 如果仅考虑单个选项的 Shapley 值, 则可能在筛选阶段遗漏对程序性能有潜在重要影响的编译选项. Shapley 交互值[36]是一个基于 Shapley 值理论的扩展概念. 在机器学习领域, Shapley 交互值能够揭示特征之间的相互依赖性, 并且衡量特征之间的相互作用对模型预测结果的影响[34]. 具体的, 对于编译选项组合 $x=(o_1, o_2,\ldots, o_n)$, 编译选项 o_i 和 o_j 的 Shapley 交互值的计算方式如下所示:

$$\phi_{i,j}=\sum_{S\subseteq O\setminus\{o_i,o_j\}}\frac{|S|!(n-|S|-2)!}{2(n-1)!}\delta_{i,j}(S),\ \text{when } i\neq j,\ \text{and } \delta_{i,j}(S)=f_x(S\cup\{o_i,o_j\})-f_x(S\cup\{o_i\})-f_x(S\cup\{o_j\})+f_x(S) \tag{8}$$

公式 (8) 中各符号的含义与 Shapley 值的计算公式基本一致, 这里不做赘述. 对于一个具体的编译调优任务, 如果两个编译选项的 Shapley 交互值很高, 那么这两个选项可能具有很强的相互作用. SHAP 框架针对树形模型同样设计了一种优化算法, 能够将计算所有特征对之间 Shapley 交互值的复杂度降低至 O($TMLD^2$).

基于 SHAP 的选项重要性分析及筛选流程如算法 1 所示. 算法首先计算验证样本集合 S_v 中每个特征 o_i 的平均 Shapley 值 $\bar{\phi}_i$, 并根据 $\bar{\phi}_i$ 对原始选项集合 O 中的选项进行排序, 筛选出排名前 50% 的选项构成新的选项集合 O', 其余选项则使用缺省配置. 这一步通过将原始选项集合 O 中对性能影响较小的选项淘汰, 从而降低了搜索空间的维度. 然后, 对于验证样本集合 S_v, 算法计算在当前模型下, 选项 $o_i\in O'$相对于其余特征 $o_j\in O$ 的平均 Shapley 交互值 $\bar{\phi}_{i,j}$. 虽然 SHAP 框架针对树形模型实现了高效的求解算法, 但是当验证样本和编译选项数量较多时, 其计算开销仍无法忽视. 针对上述问题, 一方面能够使用 SHAP 框架自带的 sample 函数对验证样本集合 S_v 进行随机采样, 以减少样本个数, 另一方面, 本文充分挖掘模块之间的潜在并行性, 在使用随机森林模型进行性能预测的同时执行基于 SHAP 的选项重要性分析和筛选, 以实现 SHAP 计算开销的隐藏, 具体实现方式如图 5 所示.

算法 1. 基于 SHAP 的选项重要性分析及筛选算法.

输入: 验证样本集合 $S_v=\{(x_k, t_k): k=1, 2,\ldots, W_v\}$, 筛选前的选项集合 O;
输出: 筛选后的选项集合 O'.

1. FOR $o_i\in O$ DO
2. FOR $(x_k, t_k)\in S_v$ DO
3. Compute $\phi_i^{(k)}$
4. END FOR
5. Compute $\bar{\phi}_i=\left(\sum_{k=1}^{W_v}|\phi_i^{(k)}|\right)/W_v$
6. END FOR
7. Sort $o_i\in O$ by $\bar{\phi}_i$
8. Reserve the top 50% elements of O as O', other elements use default value
9. FOR $o_i\in O'$ DO
10. FOR $o_i\in O$ and $i\neq j$ DO
11. FOR $(x_k, t_k)\in S_v$ DO
12. Compute $\phi_{i,j}^{(k)}$
13. END FOR
14. END FOR
15. Compute $\bar{\phi}_{i,j}=\left(\sum_{k=1}^{W_v}|\phi_{i,j}^{(k)}|\right)/W_v$
16. Use the quartile method to find potential outliers of $\bar{\phi}_{i,j}$, forming Φ
17. Compute $O'=O'\cup\{o_j\in O|\bar{\phi}_{i,j}\in\Phi\}$
18. END FOR

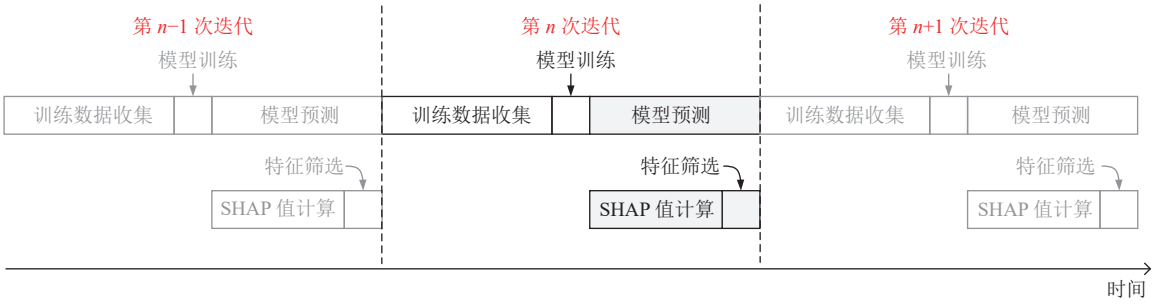


图 5 SHAP 计算开销隐藏机制示意图

最后, 对于每个选项 $o_i \in O'$, 算法使用四分位数法 (quartile method)[37]找到 $\bar{\phi}_{i,j}$ 中的所有潜在异常值, 并将所有潜在异常值对应的特征 o_j 并入 O'. 在后续的迭代中, 元搜索策略将使用筛选后的选项集合 O'构造新的搜索空间. 四分位数法是一种描述数据分布特性的统计方法, 适用于各种类型的分布, 且不受极端值的影响, 因此可以准确地反映数据集的集中趋势和离散程度. 具体地, 本文将 $\bar{\phi}_{i,j}$ 从小到大排序并分为 4 等份, Q_1 表示位于 25% 位置的数值, Q_3 表示位于 75% 位置的数值, $IQR = Q_3 - Q_1$ 表示四分位距. 在 SWTuner 的实现中, 任何高于 $Q_3 + 7.5 \times IQR$ 的值都被认为是异常值, 表明该值对应的特征 o_j 与 o_i 存在显著相关性.

为了直观地展示算法 1 的执行流程, 本文仍然使用第 2.2 节中包含 5 个选项的编译调优任务作为示例. 对于如图 4 所示的随机森林模型, 使用 SHAP 框架计算得到的平均 Shapley 值 $\bar{\phi}_i$ 以及平均 Shapley 交互值 $\bar{\phi}_{i,j}$ 如表 3 所示. 算法首先根据 $\bar{\phi}_i$ 筛选出 o_1、o_2 和 o_3 这 3 个重要选项, 然后根据 $\bar{\phi}_{i,j}$ 筛选出与 o_1 相关程度较高的 o_4 选项. 如果在选项筛选的过程中仅考虑 Shapley 值, 则 o_4 的重要性很可能会被忽视.

表 3 平均 Shapley 值以及平均 Shapley 交互值计算示例

选项	ϕ_i	$\phi_{i,j}$				
		o_1	o_2	o_3	o_4	o_5
o_1	**2.752 7**	—	0.006 5	0.009 7	**0.123 9**	0.007 0
o_2	**1.992 3**	0.006 5	—	0.009 0	0.005 8	0.002 6
o_3	**1.490 6**	0.009 7	0.009 0	—	0.006 3	0.003 1
o_4	0.257 1	0.123 9	0.005 8	0.006 3	—	0.002 5
o_5	0.249 2	0.007 0	0.002 6	0.003 1	0.002 5	—

3 实验与分析

3.1 实验环境与测试用例

本文以神威新一代超级计算机[38]为实验平台, 硬件环境主要包括前端机和计算节点. 前端机配备了 24 核心的 Intel Xeon E5-2440 处理器 (2.40 GHz), 计算节点为神威新一代众核处理器 SW26010Pro. 通过作业管理系统, SWTuner 元搜索驱动将搜索样本的编译和执行任务提交至计算节点, 而模型训练/推理以及 SHAP 计算等任务则主要在前端机上通过多进程方式并行执行. 软件环境方面, 本文选择 swGCC[39]和 swLLVM[40]两款主流编译器作为目标编译器, 以验证本文提出的 SWTuner 调优框架的兼容性. 其中, swGCC 版本为 7.1.0, swLLVM 版本为 13.0.0.

本文采用 NAS Parallel Benchmarks 3.0 (NPB 3.0)[41]作为测试用例, 以全面评估 SWTuner 的关键技术有效性以及整体调优效果. NPB 3.0 是美国国家航空航天局开发的一组基准测试程序, 共包含 8 个测试用例, 涵盖了科学计算中的多种典型算法和工作负载, 常被用来评估高性能计算系统的表现. 考虑到用例执行时间过短可能增加性能测量的相对误差, 本文将 CG、FT、IS、MG 用例的输入规模配置为 A, 其余用例配置为 W.

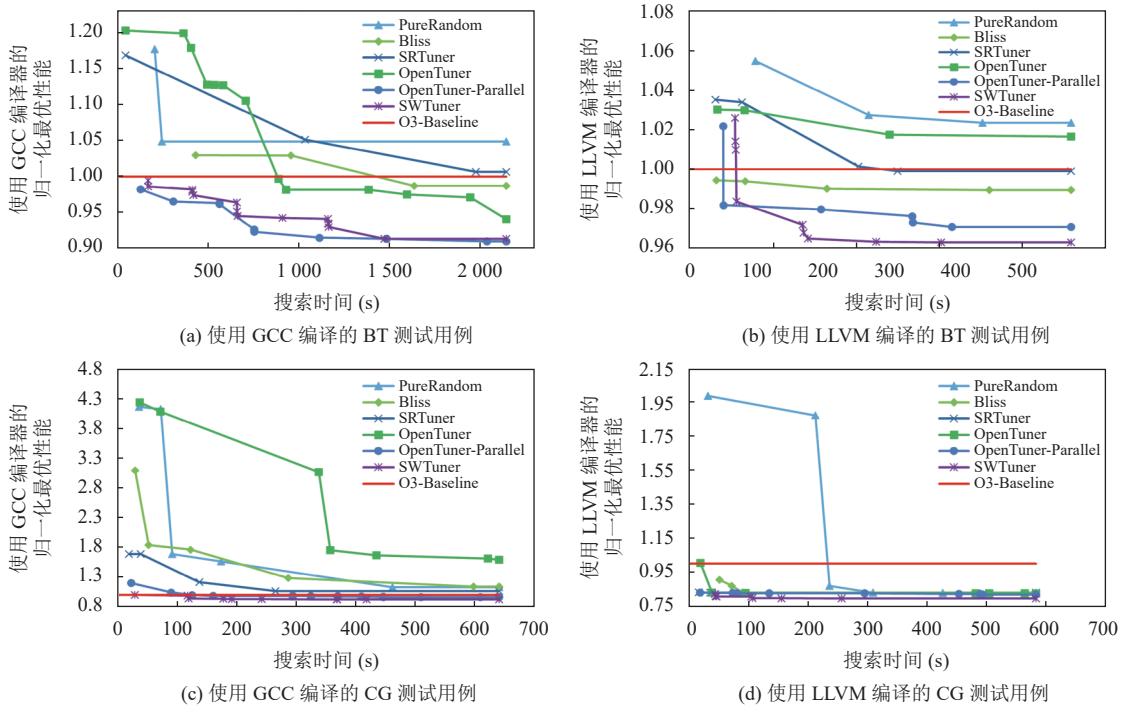
BT 和 CG 是 NPB 中的具有代表性的两个用例, 其中 BT (block tridiagonal) 是典型的计算密集型课题, 主要用

于模拟求解三对角线系统的数值方法, 其计算操作远远超过数据移动的成本. CG (conjugate gradient) 则侧重于解决大规模稀疏线性系统的问题, 具有不规则和离散的数据访问模式, 因此属于典型的访存受限课题. 在性能评测部分, 本文首先以 BT 和 CG 为例, 对关键技术的有效性进行详细分析. 然后, 本文分别展示以 swGCC 和 swLLVM 作为目标编译器的所有测试用例的性能数据, 以验证 SWTuner 调优框架的整体优化效果.

3.2 关键技术有效性评测

3.2.1 分布式元搜索

为了验证分布式元搜索策略的有效性, 本文对 PureRandom、SRTuner、Bliss、OpenTuner 和 SWTuner 在 NPB 典型测试用例上的调优效果进行了比较, 结果如图 6 所示, 其中, 横坐标为搜索时间, 纵坐标为以-O3 优化作为基准的归一化最优性能. PureRandom 是一种随机搜索方法, 通过随机选取编译选项组合的方式进行调优. SRTuner 是一个利用多阶段协同优化和分布式估计的编译器调优框架, 通过多臂老虎机模型提升调优效率. Bliss 是一个基于多样化贝叶斯优化模型池的自动调优框架, 通过动态策略与实时反馈机制实现复杂场景下的参数优化. OpenTuner 是开源的自动调优工具, 使用了一系列高效的优化算法, 如贝叶斯优化和遗传算法等, 以快速找到最优的参数设置. SWTuner 在 OpenTuner 的基础上实现了分布式元搜索策略, 能够并行收集多个搜索样本的性能数据, 以加速调优过程.



(a) 使用 GCC 编译的 BT 测试用例 (b) 使用 LLVM 编译的 BT 测试用例

(c) 使用 GCC 编译的 CG 测试用例 (d) 使用 LLVM 编译的 CG 测试用例

图 6 典型测试用例使用分布式元搜索策略的优化效果

在搜索过程的初始阶段, PureRandom 方法能够有效识别出一些具有较好性能的编译选项组合, 从而促使性能指标迅速提升. 然而, 由于搜索空间庞大, 随机搜索方法在后期容易陷入局部最优解, 从而导致搜索效率的显著下降. 与 PureRandom 方法相比, OpenTuner 采用了先进的元搜索技术和反馈机制, 能够根据先前的搜索结果动态调整后续的搜索方向, 有效避免了盲目地探索整个搜索空间. 即使在搜索过程的后期阶段, OpenTuner 也能够较为有效地识别出潜在的性能改进点, 从而确保性能提升曲线更加平缓且稳定. OpenTuner 原生支持基于 ThreadPool 的并行调优机制, 本文在神威新一代超级计算机上对其进行了适配, 24 线程并行规模下的调优效果如图 6 中 OpenTuner-Parallel 曲线所示. OpenTuner 原生的并行机制能够有效提升搜索效率, 对于 BT 和 CG 测试用例, 其在相同

时间内相比串行机制获得了约 11.9% 的平均性能提升. 然而, 基于 ThreadPool 的并行机制需要占用大量的前端机资源进行多线程编译, 一方面限制了调优任务的可扩展性, 另一方面也会影响前端机上其余的应用构建和作业管理等任务的性能.

SRTuner 和 Bliss 均采用串行搜索机制, 二者效果整体上介于 PureRandom 与 OpenTuner-Parallel 之间, 最优性能较 OpenTuner 平均提升约 6.3% 和 6.7%. SWTuner 在 OpenTuner-Parallel 的基础上实现了分布式元搜索, 通过作业管理系统调度被调优程序进行本地编译和运行, 进一步提升了可扩展性, 能够在有限的时间内探索更大的搜索空间, 这意味着它可以在相同的调优时间内发现更多的潜在优化机会. 这种分布式优化方法不仅提高了搜索效率, 还增强了搜索的全面性和准确性, 尤其是在面对复杂且庞大的搜索空间时更为明显. 从图 6 中可以看出, 相比 PureRandom 和 OpenTuner, SWTuner 的分布式元搜索策略总是能够发现更优的编译选项组合, 且对应的性能均优于-O3 基准性能, 最大提升幅度达 20.55% (使用 LLVM 编译的 CG 测试用例).

图 7 展示了分布式元搜索策略在可扩展性方面的测试结果. 在强可扩展性的评估中, 当参与计算的节点数量增加时, 各调优任务的执行时间呈现出明显的线性下降趋势. 这一结果表明通过增加计算节点的数量能够有效地提升分布式元搜索策略的处理效率. 而在弱可扩展性的测试场景下, 随着任务规模与计算节点数量的同步增长, 执行时间变化较为平缓. 这意味着在任务负载和可用资源同步增加的情况下, 分布式元搜索策略依然能够保持较好的性能稳定性.

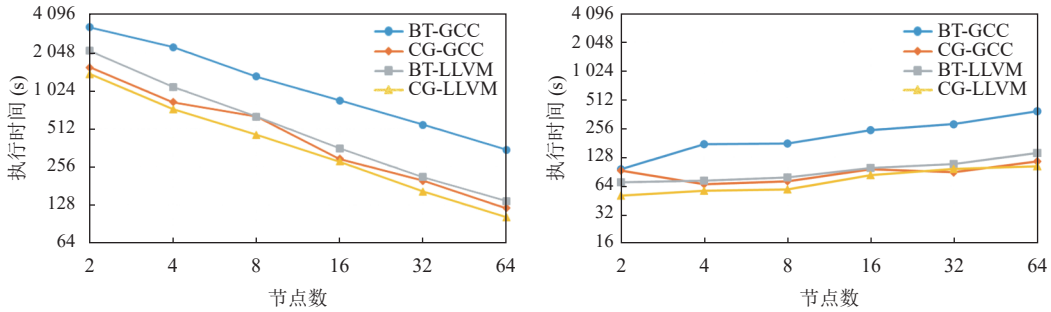


图 7 分布式元搜索策略的可扩展性测试结果

表 4 统计了分布式元搜索策略下不同搜索技术的使用次数占比情况. 其中, UGM 表示均匀贪婪变异, NGM 表示正态贪婪变异, DEM 表示差分进化变异, RNM 表示随机单纯形法. 均匀贪婪变异 (UGM) 和正态贪婪变异 (NGM) 可能因其贪婪策略在较短时间内收敛到一个较为满意的解, 因此它们在上述调优过程中更受欢迎. 差分进化变异 (DEM) 虽然具备更强的全局搜索能力, 但在某些情况下可能需要更多的迭代才能达到较好的解, 因此其使用比例相对较低. 随机单纯形法 (RNM) 作为随机方法的一种, 可能在搜索过程中引入了过多的随机性, 导致它在某些情况下不能很好地收敛到最优解. 然而, 具体选择哪种搜索方法不仅取决于算法本身的特性, 还与特定的优化目标和实际应用场景密切相关. 因此, 在不同的条件下, 各种搜索方法的有效性和适用性可能会有所不同.

表 4 分布式元搜索策略下不同搜索技术的使用次数占比 (%)

编译器	测试用例	搜索策略	Iter0	Iter1	Iter2	Iter3	平均
GCC	BT	UGM	56.50	46.20	66.70	66.70	59.03
		NGM	30.30	34.70	22.80	20.30	27.03
		DEM	12.10	18.00	10.00	12.00	13.03
		RNM	1.10	1.10	0.50	1.00	0.93
	CG	UGM	37.30	40.40	57.10	50.80	46.40
		NGM	49.40	46.60	29.40	38.90	41.08
		DEM	12.20	12.00	12.70	9.20	11.53
		RNM	1.10	1.00	0.80	1.10	1.00

表 4　分布式元搜索策略下不同搜索技术的使用次数占比 (%)(续)

编译器	测试用例	搜索策略	Iter0	Iter1	Iter2	Iter3	平均
LLVM	BT	UGM	49.60	49.50	37.10	43.50	44.93
		NGM	36.60	36.80	49.60	40.90	40.98
		DEM	13.00	12.60	12.80	15.20	13.40
		RNM	0.80	1.10	0.50	0.40	0.70
	CG	UGM	41.10	44.20	40.10	47.30	43.18
		NGM	43.30	42.90	44.30	36.60	41.78
		DEM	15.00	12.00	12.00	15.60	13.65
		RNM	0.60	0.90	0.60	0.50	0.65

3.2.2　性能建模及预测

在机器学习模型指导的性能预测阶段, 模型的训练效果直接关系到后续搜索过程的精度和效率. SWTuner 使用结果数据库中存储的历史搜索样本集合 S_t 训练一个包含 30 棵决策树的随机森林模型, 并且在每个迭代阶段计算验证样本集合 S_v 的决定系数 R^2 以评估其泛化能力 (其中 W_t/W_v=4). 如图 8 所示, 对于典型测试用例, 随机森林模型的决定系数均高于 0.96, 并且随着迭代次数的增加呈上升的趋势. 随着模型预测准确性的提高, 基于该模型的预测执行过程将能够更准确地收集性能数据, 从而优化元搜索策略, 进一步提升搜索效率.

训练成本是衡量机器学习模型实用性的关键要素之一. 深度神经网络能够学习到非常复杂的非线性关系, 但是其对训练数据的要求较高且训练开销较大, 难以适用于搜索空间频繁变化的编译调优任务. 相比之下, 随机森林模型具有较低的训练成本和更快的推理速度, 因此更适合于这类需要快速迭代的应用场景. 如图 9 所示, 随机森林模型在确保预测精度达到较高水平的同时, 最大训练时间开销仅为 0.4 s 左右. 值得注意的是, 随着迭代次数的增加, 模型的整体训练时间呈现出逐渐减少的趋势, 这意味着模型的学习效率在不断提升, 优化过程逐渐成熟. 上述数据表明, 随机森林模型在处理大规模数据集时具有良好的收敛性和稳定性, 为编译调优任务的快速迭代和实时决策提供了有力支持.

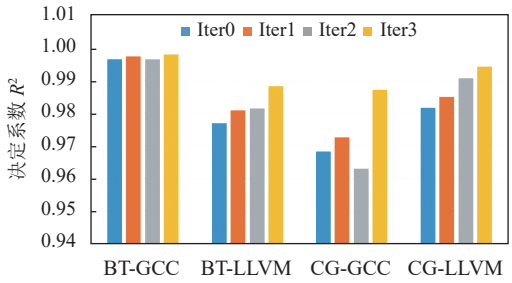


图 8　典型测试用例在不同迭代中的模型训练效果

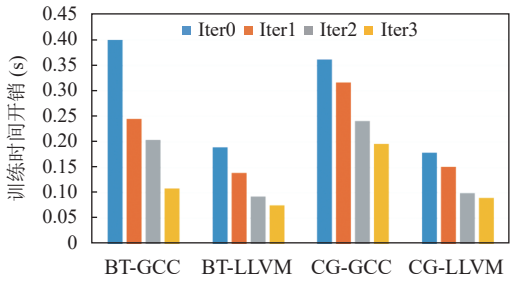


图 9　典型测试用例在不同迭代中的模型训练开销

较高的预测准确性以及较低的训练开销使得通过性能预测替代部分实际执行成为可能, 这种方法能够在较大程度上帮助减少系统的整体功耗. 性能预测阶段的样本集合记为 $S_p=\{(x_i, t_i, t_i'): i=1, 2,\ldots, W_p\}$, 其中 t_i 表示选项组合 x_i 的预测执行时间, t_i' 表示实际执行时间. 假设调优过程中测试用例在单位时间内的运行功耗为固定值, 则相较于传统的实际运行方式, SWTuner 在性能预测阶段的功耗节省比例可通过公式 (9) 近似计算获得:

$$Ratio_{\text{PowerSaving}} \approx \frac{\sum_{((x_i,t_i,t_i')\in S_p)\cap(t_i\leqslant ET)} t_i}{\sum_{((x_i,t_i,t_i')\in S_p)\cap(t_i\leqslant ET)} t_i + \sum_{((x_i,t_i,t_i')\in S_p)\cap(t_i>ET)} t_i'} \tag{9}$$

图 10 展示了 NPB 典型测试用例在性能预测阶段的功耗节省情况, 所有典型用例的平均功耗节省比例约为 23.2%. 从图 10 中可以看出, 编译器类型对功耗节省的影响较为显著, GCC 和 LLVM 编译器的平均功耗节省比例

分别为 34.11% 和 12.28%. 出现该现象的主要原因是 LLVM 编译器的大部分优化选项对程序性能的影响较小, 使得程序性能分布较为集中, 而搜索过程中多数样本的性能位于均值左侧, 从而导致较为频繁的实际执行.

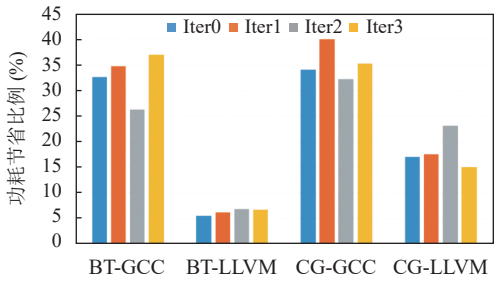


图 10　典型测试用例在性能预测阶段的功耗节省比例

3.2.3　选项分析及筛选

编译选项数量的增长导致了搜索空间的指数级膨胀, 使得遍历寻找最优配置变得不切实际. 主流编译调优框架采用启发式方法寻求近似最优解, 但需在优化质量和效率间权衡. 本文提出的基于 SHAP 的选项分析及筛选方法综合考虑编译选项的重要程度及依赖关系, 通过逐步减小搜索空间的方式保证搜索过程聚焦于对性能调优有显著影响的选项集合. 图 11 以 GCC 编译器编译的 BT 测试用例为例, 展示了 SWTuner 在不同迭代阶段中的最优性能变化情况, 并且与 BOCA 框架的调优效果进行了对比, 其中纵坐标为以-O3 优化作为基准的归一化最优执行时间. 对于 SWTuner 框架, 在单次迭代内部, 程序的执行时间均呈现下降的趋势. 然而, 随着搜索时间的增加, 搜索算法可能会陷入局部最优解, 导致性能的变化趋于稳定而难以找到全局最优解. 经过 SWTuner 的 4 轮迭代调优, BT 测试用例的归一化最优执行时间最终稳定在 0.9 左右, 相较于前 3 轮迭代的最优性能分别提升 1.2%、2.7% 及 0.4%.

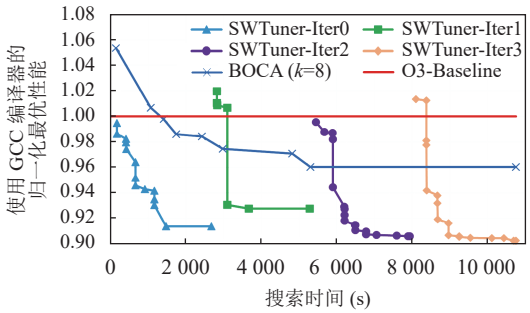


图 11　BT 测试用例在不同迭代中的最优性能变化情况

为了降低搜索空间的维度, BOCA 框架在每轮迭代中选择重要性最高的 k 个选项进行变异, 图 11 中给出了 k=8 情况下最优性能的变化情况. 如图所示, BOCA 在调优初始阶段能够获得与 SWTuner 相当的收敛速度 (介于 Iter1 与 Iter2 之间), 但后期调优效果显著下降, 最优性能仅为 SWTuner 第 4 轮迭代的约 93.9%. BOCA 框架默认采用平均不纯度减少法 (mean decrease impurity, MDI) 评估选项的重要性, 其计算高效但可能高估某些孤立的重要选项. 相较于 MDI 方法, 基于 SHAP 的选项分析及筛选方法能够显式建模选项间的交互效应, 从而更加精准地识别关键选项及其组合, 推动调优效果持续改进.

正如第 2.3 节中算法 1 所述, 为了确保选项筛选的有效性, SWTuner 运用 Shapley 值理论来量化分析编译选项与程序性能之间的关系, 并从中筛选出对程序性能有显著影响的选项集合. 表 5 统计了上述 BT 示例在各个迭代阶段中重要性排名前 10 的编译选项及其对应的 Shapley 值, 并通过颜色来区分选项出现的频率: 出现 4 次的选项被标记为红色, 出现 3 次的选项被标记为绿色, 而出现 2 次的选项则被标记为蓝色. 具体而言, -ffloat-store、-Ox 和 -fschedule-insns 这 3 个选项在整个迭代过程中始终展现出较高的 Shapley 值, 表明其对于 BT 测试用例具有显著

的重要性, 这一结果也与我们对于程序特性和选项功能的理解相一致. 此外, 随着迭代次数的增加, 排名前 10 的编译选项中被标记颜色的比例也随之提高. 特别地, 在第 4 次迭代 (Iter3) 中, 排名前 10 的编译选项均曾在之前的迭代中被标记. 这表明基于 SHAP 的选项分析及筛选方法具有良好的稳定性, 能够有效地识别并保留对优化目标产生显著影响的编译选项.

表 5　BT 测试用例在不同迭代中排名前 10 的 GCC 编译选项及其对应的 Shapley 值

迭代	BT测例编译选项及对应的Shapley值	Others
Iter0	-ffloat-store (9.088), -Ox (5.290), -fschedule-insns (0.904), -ftree-loop-optimize (0.082), -ftree-ch (0.078), -fif-conversion2 (0.039), -fsched-interblock (0.035), -fipa-pta (0.035), -fmove-loop-invariants (0.034), -fdevirtualize-speculatively (0.034)	181 other flags
Iter1	-ffloat-store (8.506), -Ox (6.877), -fschedule-insns (0.671), -ftree-forwprop (0.212), -fnon-call-exceptions (0.121), -fselective-scheduling (0.066), -fipa-pta (0.060), -fsw-veclib (0.053), -fschedule-insns2 (0.053), -fipa-cp (0.053)	119 other flags
Iter2	-ffloat-store (8.483), -Ox (4.420), -fschedule-insns (0.831), -ftree-loop-optimize (0.293), -ftree-forwprop (0.093), -fselective-scheduling (0.090), -flive-range-shrinkage (0.070), -fschedule-fusion (0.050), -fipa-pta (0.048), -ftrapv (0.042)	60 other flags
Iter3	-ffloat-store (9.559), -Ox (5.528), -fschedule-insns (0.917), -fselective-scheduling (0.207), -ftrapv (0.158), -ftree-forwprop (0.097), -ftree-loop-optimize (0.073), -flive-range-shrinkage (0.059), -fipa-pta (0.057), -ftree-ch (0.035)	26 other flags

考虑到求解 Shapley 值及其交互值的复杂度较高, 选项重要性分析的时间开销不容忽视. 如图 12 和图 13 所示, 对于随机森林模型, Shapley 值及 Shapley 交互值的计算开销均随着迭代次数的增加而呈现出下降趋势. 具体而言, Shapley 值的计算开销整体较低, 在当前示例中最高仅为 1.65 s. 由于 Shapley 值的计算在每个迭代阶段仅需执行一次, 因此相对于整个调优过程而言可忽略不计. Shapley 交互值的计算复杂度与选项数量线性相关, 因此不同编译器的计算开销存在差异. 对于上述示例, GCC 和 LLVM 的平均开销分别为 230.88 s 和 85.16 s, 与各编译器的选项数量情况基本保持一致. 通过在性能预测的同时执行选项重要性分析和筛选操作, SWTuner 能够有效地隐藏上述计算开销, 从而实现高效的自动调优.

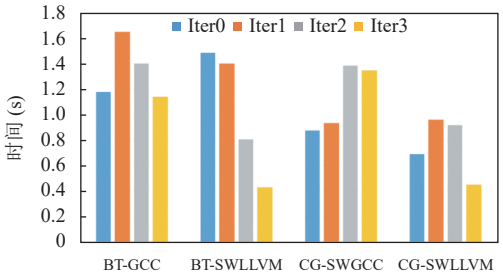


图 12　典型测试用例在不同迭代中的 Shapley 值计算开销

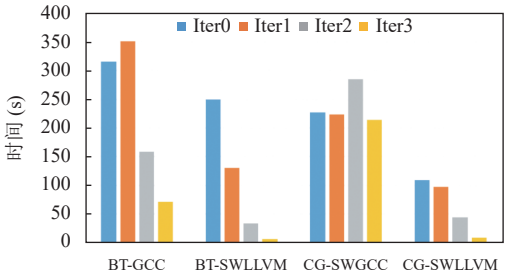


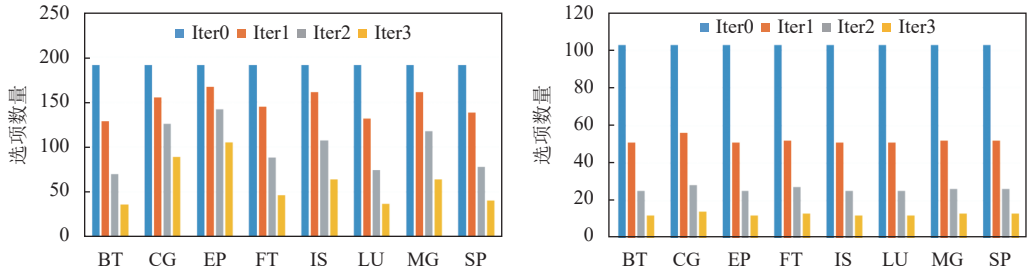
图 13　典型测试用例在不同迭代中的 Shapley 交互值计算开销

3.3　整体优化效果评测

为了验证 SWTuner 的整体调优效果, 本文使用 GCC 和 LLVM 编译器分别对 NPB 测试集中的所有测试用例进行了 4 轮迭代调优, 图 14 展示了调优过程中编译选项数量的变化情况. 对于所有测试用例, 编译选项的数量均随着迭代次数的增加而下降, 这表明基于 SHAP 的选项分析及筛选算法能够有效地降低搜索空间的维度. 同时, 与 LLVM 编译器相比, GCC 编译器的选项数量随迭代次数的变化较为缓慢. 这意味着在每轮迭代的筛选阶段之后, 有较多的选项被重新添加到集合 O'中, 以避免遗漏对程序性能有潜在重要影响的编译选项. 这一现象表明, GCC 编译器的选项之间可能存在着更强的相关性和依赖性, 导致即使经过筛选后, 也需要保留更多的选项以维持优化的有效性.

相应地, 本文以-O3 优化作为基准, 记录了调优过程中各个用例的归一化最优执行时间的变化情况. 如表 6 所示 (每个测试用例在所有迭代阶段获得的全局最优性能用红色粗体标记), 随着调优过程中编译选项数量的不断减

少, 各测试用例的归一化最优执行时间均保持在较为稳定的水平, 并且大多数测试用例经过迭代后均获得了性能提升. 与仅使用-O3 优化相比, 加速比最高达到了 1.25 (在使用 LLVM 编译器处理 CG 测试用例的第 3 轮迭代中), 这进一步证明了基于 SHAP 的选项分析及筛选算法的稳定性和有效性. 同时, 不同编译器的调优效果表现出较为明显的差异: GCC 编译器在不同迭代间显示出较大的性能波动, 而 LLVM 编译器则表现得更加稳定. 这种现象可能是由于编译器的设计理念和技术实现不同所导致的. 具体来说, GCC 可能采用了更多依赖于特定上下文的优化技术, 导致在不同迭代中的性能结果差异较大; 而 LLVM 可能使用了更为独立且通用的优化方法, 因此在迭代间维持了更高的性能一致性. 特别地, CG 测试用例使用 LLVM 编译器的调优效果 (20.63%) 显著优于 GCC 编译器 (8.05%). 通过分析选项筛选过程能够发现, -ffast-math 选项在 LLVM 编译器的多轮迭代中始终保持较高的 Shapley 值. 对于 swLLVM 编译器, -ffast-math 是一个需要显式启用的集合选项, 其通过放宽严格的 IEEE-754 浮点运算规则限制, 允许编译器进行一系列激进的优化, 包括循环展开、指令重排、乘加融合等. 而在当前的 swGCC 编译器实现中, -O3 基准优化级别已启用了部分浮点运算优化, 因此受-ffast-math 选项的影响较小.



(a) GCC 编译选项　　(b) LLVM 编译选项

图 14　NPB 测试集在调优过程的不同迭代阶段应用的编译选项数量

表 6　NPB 测试集调优过程中不同迭代阶段的最优归一化执行时间 (以-O3 为基准)

测试用例	GCC归一化最优执行时间				LLVM归一化最优执行时间			
	Iter0	Iter1	Iter2	Iter3	Iter0	Iter1	Iter2	Iter3
BT	0.9136	0.9276	0.9058	**0.9024**	0.9632	0.9606	**0.9592**	0.9599
CG	**0.9195**	0.9438	0.9685	0.9378	0.7946	0.7941	**0.7937**	0.7938
EP	**0.9569**	0.9599	0.9602	0.9595	0.9856	0.9856	0.9852	**0.9852**
FT	0.9216	0.9883	0.9299	**0.9022**	0.9063	0.9066	**0.9058**	0.9063
IS	0.8123	0.8124	0.8139	**0.8118**	0.9910	0.9910	**0.9904**	0.9906
LU	0.9676	0.9483	**0.9264**	0.9302	0.9864	0.9859	0.9859	**0.9834**
MG	**0.9540**	0.9596	0.9679	0.9658	0.9142	0.9137	0.9146	**0.9135**
SP	0.9692	**0.9577**	0.9626	0.9635	0.9874	**0.9852**	0.9879	0.9880

4　结束语

本文设计并实现了一款名为 SWTuner 的新型编译调优框架, 该框架整合了分布式元搜索策略与先进的机器学习技术, 旨在解决传统编译调优方法在处理复杂优化选项组合时所面临的局限性. SWTuner 通过引入 AUC-Bandit 分布式元搜索策略, 有效地降低了编译和运行过程的时间开销, 从而提高了搜索效率. 此外, SWTuner 利用随机森林模型对编译选项与程序性能之间的关系进行了建模, 并将其应用于元搜索过程中, 从而减少了实际执行的功耗开销. 通过引入 Shapley 值及其交互值来评估编译选项的重要程度及相关性, SWTuner 能够在确保调优效果的同时, 显著降低搜索空间的维度. 实验结果表明, 在神威新一代超级计算机上对 NPB 基准测试用例进行的调优中, 相比于其他主流调优方法或工具, SWTuner 在提升搜索效率的同时显著降低了功耗开销. 通过分析和筛选对程序性能影响显著的选项, SWTuner 还能够进一步地指导应用算法及编译优化的改进.

未来的研究将深入探讨 SWTuner 在更大规模并行计算环境中的应用, 特别是在异构计算平台上的性能表现. 同时, 本研究将致力于提高模型的预测精度, 特别是在处理新型应用或架构时, 增强模型的泛化能力. 此外, 还将探索更高效的搜索策略, 以更好地平衡探索与利用之间的关系, 避免陷入局部最优解. 最后, 计划将 SWTuner 扩展至支持更多的编译器和编程模型, 使其成为一个更为广泛适用的自动编译调优工具.

References

[1] Bacon DF, Graham SL, Sharp OJ. Compiler transformations for high-performance computing. ACM Computing Surveys (CSUR), 1994, 26(4): 345–420. [doi: 10.1145/197405.197406]

[2] Sarkar V. Optimized unrolling of nested loops. In: Proc. of the 14th Int'l Conf. on Supercomputing. New Mexico: ACM, 2000. 153–166. [doi: 10.1145/335231.335246]

[3] Wegman MN, Zadeck FK. Constant propagation with conditional branches. ACM Trans. on Programming Languages and Systems (TOPLAS), 1991, 13(2): 181–210. [doi: 10.1145/103135.103136]

[4] Golovanevsky O, Zaks A. Struct-reorg: Current status and future perspectives. In: Proc. of the 2007 GCC Developers' Summit. 2007. 47–56.

[5] Ayers A, Schooler R, Gottlieb R. Aggressive inlining. In: Proc. of the 1997 ACM SIGPLAN Conf. on Programming Language Design and Implementation. Las Vegas: ACM, 1997. 134–145. [doi: 10.1145/258915.258928]

[6] Nuzman D, Rosen I, Zaks A. Auto-vectorization of interleaved data for SIMD. ACM SIGPLAN Notices, 2006, 41(6): 132–143. [doi: 10.1145/1133255.1133997]

[7] Dagum L, Menon R. OpenMP: An industry standard API for shared-memory programming. IEEE Computational Science and Engineering, 1998, 5(1): 46–55. [doi: 10.1109/99.660313]

[8] Ryoo S, Rodrigues CI, Baghsorkhi SS, Stone SS, Kirk DB, Hwu WMW. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In: Proc. of the 13th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. Salt Lake City: ACM, 2008. 73–82. [doi: 10.1145/1345206.1345220]

[9] Chung IH, Hollingsworth JK. A case study using automatic performance tuning for large-scale scientific programs. In: Proc. of the 15th IEEE Int'l Conf. on High Performance Distributed Computing. Paris: IEEE, 2006. 45–56. [doi: 10.1109/HPDC.2006.1652135]

[10] Reinders J, Ashbaugh B, Brodman J, Kinsner M, Pennycook J, Tian XM. Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems Using C++ and SYCL. New York: Apress Berkeley, 2021. [doi: 10.1007/978-1-4842-5574-2]

[11] Evans AN, Campbell B, Soffa ML. Is rust used safely by software developers? In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 246–257. [doi: 10.1145/3377811.3380413]

[12] Bezanson J, Edelman A, Karpinski S, Shah VB. Julia: A fresh approach to numerical computing. SIAM Review, 2017, 59(1): 65–98. [doi: 10.1137/141000671]

[13] Ashouri AH, Killian W, Cavazos J, Palermo G, Silvano C. A survey on compiler autotuning using machine learning. ACM Computing Surveys (CSUR), 2018, 51(5): 96. [doi: 10.1145/3197978]

[14] Bodin F, Kisuki T, Knijnenburg P, O'Boyle M, Rohou E. Iterative compilation in a non-linear optimisation space. 1998. https://liacs.leidenuniv.nl/assets/PDF/TechRep/tr98-16.pdf

[15] Bei ZD, Yu ZB, Zhang HL, Xiong W, Xu CZ, Eeckhout L, Feng SZ. RFHOC: A random-forest approach to auto-tuning Hadoop's configuration. IEEE Trans. on Parallel and Distributed Systems, 2016, 27(5): 1470–1483. [doi: 10.1109/TPDS.2015.2449299]

[16] Chen TQ, Moreau T, Jiang ZH, Zheng LM, Yan E, CowanM, Shen HC, Wang LY, Hu YW, Ceze L, Guestrin C, Krishnamurthy A. TVM: An automated end-to-end optimizing compiler for deep learning. In: Proc. of the 13th USENIX Conf. on Operating Systems Design and Implementation. Carlsbad: USENIX Association, 2018. 579–594.

[17] Mahgoub A, Wood P, Ganesh S, Mitra S, Gerlach W, Harrison T, Meyer F, Grama A, Bagchi S, Chaterji S. Rafiki: A middleware for parameter tuning of NoSQL datastores for dynamic metagenomics workloads. In: Proc. of the 18th ACM/IFIP/USENIX Middleware Conf. Las Vegas: ACM, 2017. 28–40. [doi: 10.1145/3135974.3135991]

[18] Ashouri A, Bignoli A, Palermo G, Silvano C, Kulkarni S, Cavazos J. MiCOMP: Mitigating the compiler phase-ordering problem using optimization sub-sequences and machine learning. ACM Trans. on Architecture and Code Optimization (TACO), 2017, 14(3): 29. [doi: 10.1145/3124452]

[19] Liu HZ, Luo J, Li Y, Wu ZH. Iterative compilation optimization based on metric learning and collaborative filtering. ACM Trans. on Architecture and Code Optimization (TACO), 2021, 19(1): 2. [doi: 10.1145/3480250]

[20] Theodoridis T, Grosser T, Su ZD. Understanding and exploiting optimal function inlining. In: Proc. of the 27th ACM Int'l Conf. on

Architectural Support for Programming Languages and Operating Systems. Lausanne: ACM, 2022. 977–989. [DOI: 10.1145/3503222.3507744]

[21] Ashouri AH, Elhoushi M, Hua YZ, Wang X, Manzoor MA, Chan B, Gao Y. Work-in-progress: MLGOPerf: An ML guided inliner to optimize performance. In: Proc. of the 2022 Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Systems. Shanghai: IEEE, 2022. 3–4. [doi: 10.1109/CASES55004.2022.00008]

[22] Li MH, Zhang MJ, Wang C, Li MQ. AdaTune: Adaptive tensor program compilation made efficient. In: Proc. of the 34th Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2020. 1241.

[23] Zheng LM, Jia CF, Sun MM, Wu Z, Yu CH, Haj-Ali A, Wang Y, Yang J, Zhuo DY, Sen K, Gonzalez JE, Stoica I. Ansor: Generating high-performance tensor programs for deep learning. In: Proc. of the 14th USENIX Symp. on Operating Systems Design and Implementation. USENIX Association, 2020. 863–879.

[24] Park S, Latifi S, Park Y, Behroozi A, Jeon B, Mahlke S. SRTuner: Effective compiler optimization customization by exposing synergistic relations. In: Proc. of the 2022 IEEE/ACM Int'l Symp. on Code Generation and Optimization. Seoul: IEEE, 2022. 118–130. [doi: 10.1109/CGO53902.2022.9741263]

[25] Chen JJ, Xu NX, Chen PQ, Zhang HY. Efficient compiler autotuning via Bayesian optimization. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 1198–1209. [doi: 10.1109/ICSE43902.2021.00110]

[26] Roy RB, Patel T, Gadepally V, Tiwari D. Bliss: Auto-tuning complex applications using a pool of diverse lightweight learning models. In: Proc. of the 42nd ACM SIGPLAN Int'l Conf. on Programming Language Design and Implementation. ACM, 2021. 1280–1295. [doi: 10.1145/3453483.3454109]

[27] Ansel J, Kamil S, Veeramachaneni K, Ragan-Kelley J, Bosboom J, O'Reilly UM, Amarasinghe S. OpenTuner: An extensible framework for program autotuning. In: Proc. of the 23rd Int'l Conf. on Parallel Architecture and Compilation Techniques. Edmonton: IEEE, 2014. 303–315. [doi: 10.1145/2628071.2628092]

[28] Wang ZY, Tang YH, Chen J, Xue JL, Zhou Y, Dong Y. Energy wall for exascale supercomputing. Computing and Informatics, 2016, 35(4): 941–962.

[29] Balaprakash P, Dongarra J, Gamblin T, Hall M, Hollingsworth JK, Norris B, Vuduc R. Autotuning in high-performance computing applications. Proc. of the IEEE, 2018, 106(11): 2068–2083. [doi: 10.1109/JPROC.2018.2841200]

[30] Breiman L. Random forests. Machine Learning, 2001, 45(1): 5–32. [doi: 10.1023/A:1010933404324]

[31] Ozer DJ. Correlation and the coefficient of determination. Psychological Bulletin, 1985, 97(2): 307–315. [doi: 10.1037/0033-2909.97.2.307]

[32] Kwak SG, Kim JH. Central limit theorem: The cornerstone of modern statistics. Korean Journal of Anesthesiology, 2017, 70(2): 144–156. [doi: 10.4097/kjae.2017.70.2.144]

[33] Lipovetsky S, Conklin M. Analysis of regression in game theory approach. Applied Stochastic Models in Business and Industry, 2001, 17(4): 319–330. [doi: 10.1002/asmb.446]

[34] Molnar C, Casalicchio G, Bischl B. Interpretable machine learning-a brief history, state-of-the-art and challenges. In: Proc. of the 2020 Workshops of Joint European Conf. on Machine Learning and Knowledge Discovery in Databases. Ghent: Springer, 2020. 417–431. [doi: 10.1007/978-3-030-65965-3_28]

[35] Lundberg SM, Lee SI. A unified approach to interpreting model predictions. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 4768–4777.

[36] Fujimoto K, Kojadinovic I, Marichal JL. Axiomatic characterizations of probabilistic and cardinal-probabilistic interaction indices. Games and Economic Behavior, 2006, 55(1): 72–99. [doi: 10.1016/j.geb.2005.03.002]

[37] Wan X, Wang WQ, Liu JM, Tong TJ. Estimating the sample mean and standard deviation from the sample size, median, range and/or interquartile range. BMC Medical Research Methodology, 2014, 14: 135. [doi: 10.1186/1471-2288-14-135]

[38] Liu Y, Liu X, Li F, Fu HH, Yang YL, Song JW, Zhao PP, Wang Z, Peng DJ, Chen HR, Guo C, Huang HL, Wu WZ, Chen DX. Closing the "quantum supremacy" gap: Achieving real-time simulation of a random quantum circuit using a new Sunway supercomputer. In: Proc. of the 2021 Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. St. Louis: IEEE, 2021. 1–12. [doi: 10.1145/3458817.3487399]

[39] Wu W, Qian H, Zhu Q, Wang J, Fan XJ. Research on full-chip programming for sunway heterogeneous many-core processor. In: Proc. of the 3rd World Symp. on Software Engineering. Xiamen: ACM, 2021. 174–179. [doi: 10.1145/3488838.3488868]

[40] Shen L, Zhou WH, Wang F, Xiao Q, Wu WH, Zhang LF, An H, Qi FB. swLLVM: Optimized compiler for new generation Sunway supercomputer. Ruan Jian Xue Bao/Journal of Software, 2024, 35(5): 2359–2378 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6896.htm [doi: 10.13328/j.cnki.jos.006896]

[41] NPB3.0-omp-C. 2024. https://github.com/benchmark-subsetting/NPB3.0-omp-C

附中文参考文献

[40] 沈莉, 周文浩, 王飞, 肖谦, 武文浩, 张鲁飞, 安虹, 漆锋滨. swLLVM: 面向神威新一代超级计算机的优化编译器. 软件学报, 2024, 35(5): 2359–2379. http://www.jos.org.cn/1000-9825/6896.htm [doi: 10.13328/j.cnki.jos.006896]

作者简介

周文浩, 博士生, 主要研究领域为异构众核编程模型, 编译系统.
沈莉, 博士生, 主要研究领域为编译系统, 编译优化.
王飞, 博士生, 主要研究领域为编译优化, 众核编程环境.
肖谦, 博士生, 主要研究领域为编译优化, 数据流计算, AI 框架.
李斌, 工程师, 主要研究领域为异构众核编译系统.
高秀武, 博士, 助理研究员, 主要研究领域为异构众核编译系统.
宋长明, 博士生, 主要研究领域为高性能计算系统软件.
安虹, 博士, 教授, CCF 高级会员, 主要研究领域为并行计算系统, 众核芯片架构.
漆锋滨, 博士, 正高级工程师, CCF 会士, 主要研究领域为高性能计算体系结构, 编译优化.