

自动调优与反馈导向优化融合方法研究

刘秉鑫, 黄颖晖, 高建花, 石剑君, 计卫星*

(北京师范大学人工智能学院, 北京 100875)

摘 要: 编译器自动调优与反馈导向优化(Feedback-Directed Optimization, FDO)是提升应用程序性能的两大关键技术。自动调优技术通过在高维离散编译选项空间中迭代搜索, 获得较优配置, 但其反馈信号通常仅依赖程序运行时间等指标, 缺乏对程序动态运行时信息的利用能力; 而反馈导向优化通过插桩或采样方式收集程序运行时剖面信息, 并用以指导内联、代码布局与循环优化等关键优化决策。然而FDO高度依赖运行时剖面信息的质量, 且受限于静态优化信息, 在优化选项变化时使剖面信息失效从而导致负优化。本文在统一搜索预算约束下, 将自动调优与FDO深度融合, 提出一种基于运行时剖面更新策略的联合优化框架。该框架将编译选项搜索、程序编译执行、运行时剖面采集以及反馈导向优化过程统一建构为一个完整的优化系统, 并引入剖面来源配置等中间状态变量, 以刻画剖面信息与当前候选编译配置之间的一致性关系。在此基础上, 本文设计了三种融合模式: 两阶段分离搜索、固定间隔采样搜索以及条件触发采样搜索。其中, 两阶段方法将搜索空间分解为非FDO选项与FDO选项分别优化; 固定间隔采样通过周期性更新剖面以保持剖面信息的有效性; 条件采样则基于编译选项变化程度与风险划分动态决定是否触发剖面更新, 从而在开销与信息有效性之间取得平衡。进一步地, 本文提出基于结构破坏风险的选项分类方法, 将FDO相关优化选项划分为高风险、低风险与无风险三类, 并通过Hamming距离刻画候选配置与剖面信息来源配置之间的偏离程度, 以实现自适应剖面更新机制。该方法能够有效减少不必要的剖面采集开销, 同时避免剖面信息失效导致的优化误导。实验基于GCC 12.2与OpenTuner框架, 在PolyBench测试集上进行评估。结果表明, 相较于-O3优化基线, 条件采样方法可达到1.184x的几何平均加速比, 相较于标准OpenTuner提升约8.46%, 同时在收敛速度与鲁棒性方面表现更优。

关键词: 代码优化; 自动调优; 反馈导向优化; 条件采样; 编译优化; 搜索算法

基金项目: 国家自然科学基金(No.62372046)

中图分类号: TP319

文献标识码: A

文章编号: 0372-2112(2026)07-0248-14

电子学报 URL: <http://www.ejournal.org.cn>

DOI: 10.12263/DZXB.20260384

An Integration Method for Autotuning and Feedback-Directed Optimization

LIU Bingxin, HUANG Yinghui, GAO Jianhua, SHI Jianjun, JI Weixing*

(School of Artificial Intelligence, Beijing Normal University, Beijing 100875, China)

Abstract: Compiler autotuning and feedback-directed optimization (FDO) are two key techniques for improving application performance. Autotuning obtains superior configurations through iterative search in a high-dimensional discrete compiler option space. However, its feedback signals usually rely only on metrics such as program execution time, and thus lack the capability to exploit dynamic runtime information. In contrast, FDO collects runtime profile information through instrumentation or sampling and uses it to guide key optimization decisions, such as inlining, code layout, and loop optimization. Nevertheless, FDO heavily depends on the quality of runtime profile information and is constrained by static optimization information. When optimization options change, profile information may become invalid, which may in turn lead to performance degradation. Under a unified search budget constraint, this paper deeply integrates autotuning with FDO and proposes a joint optimization framework based on a runtime profile update strategy. The framework formulates compiler option search, program compilation and execution, runtime profile collection, and feedback-directed optimization as an integrated optimization system. It further introduces intermediate state variables, such as the profile source configuration, to characterize the consistency relationship between profile information and the current candidate compilation configuration. On this basis, three integration modes are designed: two-stage decoupled search, fixed-interval sampling search, and condition-triggered sampling search. The two-stage method decomposes the search space into non-FDO options and FDO-related options and optimizes them separately. Fixed-interval sampling periodically updates the profile to maintain the validity of

profile information. Conditional sampling dynamically determines whether to trigger profile updates based on the degree of compiler-option changes and risk classification, thereby striking a balance between overhead and information validity. Furthermore, this paper proposes an option classification method based on structural-disruption risk, which divides FDO-related optimization options into high-risk, low-risk, and no-risk categories. The Hamming distance is used to quantify the deviation between a candidate configuration and the profile source configuration, enabling an adaptive profile update mechanism. The proposed method can effectively reduce unnecessary profile collection overhead while avoiding optimization misguidance caused by invalid profile information. Experiments are conducted on the PolyBench benchmark suite using GCC 12.2 and the OpenTuner framework. The results show that, compared with the -O3 optimization baseline, the conditional sampling method achieves a geometric mean speedup of 1.184 $\times$ and improves performance by approximately 8.46% over standard OpenTuner, while also exhibiting better convergence speed and robustness.

Keywords: code optimization; autotuning; feedback-directed optimization; conditional sampling; compiler optimization; search algorithm

Foundation Item(s): National Natural Science Foundation of China (No.62372046)

0 引言

现代通用编译器包含大量相互耦合的优化遍,其默认优化策略难以在不同程序、不同输入甚至同一程序的不同函数之间同时保持最优。编译优化选项的重排与选择因此成为研究热点问题之一^[1-3]。已有工作提出了以真实运行指标驱动的自动调优(Autotuning)技术,即将可调编译配置显式化,并通过多次编译、运行测试,以在较大的配置空间中发现比-O2和-O3等更优的编译配置,从而突破静态启发式优化器依赖人工经验的局限性^[3]。与此同时,反馈导向优化,又称剖面信息引导优化(Profile-Guided Optimization, PGO),使用有代表性的输入收集程序运行时的剖面信息,例如程序热点、执行边频率、分支行为等,并将运行时剖面信息反馈给编译器以改进关键优化决策,如代码布局与内联决策等,从而在程序真实输入上获得性能收益^[4]。

基于迭代搜索的Autotuning技术通过改变编译选项,可以获得显著优于固定优化选项的性能表现,但是在实际应用中仍有不足。一方面,自动调优工具需要大量编译和执行才能获得稳定的反馈指标,从而导致大规模程序的搜索成本较高^[3,5-7];另一方面,迭代搜索过程虽多次运行程序,但并没有利用程序的动态执行信息指导编译优化过程。相比之下,FDO技术通过显式引入运行时捕获的信息,为编译优化器提供了更细粒度、更贴近程序运行中真实行为的描述,从而提高优化决策依据的真实性和可靠性^[8-12]。目前虽已经提出多种基于剖面信息的反馈优化技术,但是并非所有的优化都会带来正优化,针对不同的程序开展高效、细粒度、优化效果可保证的反馈优化成为一个亟待解决的问题。

Autotuning与FDO各自特点具有天然的互补性。Autotuning擅长在大规模静态配置空间中自动探索更好的编译方案,而FDO则聚焦于利用从真实运行过

程中获得的运行时剖面信息提升关键优化决策的精度。然而,当Autotuning与FDO结合时,运行时剖面信息的角色发生变化:其既是FDO优化器的输入,又是自动调优搜索过程中由某一候选配置采集得到的中间状态。若自动调优搜索算法生成的后续候选配置改变影响了程序调用图、控制流或数据流,原有运行时剖面信息可能无法准确映射或者不再代表其真实热点行为。因此,二者融合的关键是在有限预算下决定何时采集新剖面、何时复用旧剖面,以及如何将剖面来源配置与当前优化配置的关系纳入搜索过程。基于上述背景,本文探讨Autotuning与FDO的融合优化方法,在保持已有编译器工作流不变的前提下,将编译选项搜索、运行时间获取、剖面信息收集、反馈导向编译优化纳入统一的优化框架。本文在将Autotuning与FDO技术进行整合的同时,将候选编译配置生成、程序编译与运行、运行时剖面信息采集、反馈导向编译优化、性能评估整个过程视为统一预算约束下的联合优化过程。其中,Autotuning负责候选配置的生成、搜索预算的控制以及性能指标的优化,FDO作为动态的运行时反馈信息嵌入Autotuning搜索迭代循环,并解决运行时剖面信息何时进行再次采样更新的额外决策问题。围绕上述问题,本文的主要贡献如下:

(1)提出Autotuning与FDO统一融合建模方法,将候选优化配置、剖面来源配置与剖面更新策略纳入统一预算约束下的搜索循环,在该统一框架下系统化设计两阶段分离搜索、固定间隔采样搜索和条件采样搜索三类融合模式,比较不同剖面更新策略对性能和开销的影响;

(2)提出基于结构风险划分与配置偏离度量的条件采样策略,在高风险结构变化时及时更新剖面,在低风险或无关变化时复用已有剖面,从而降低剖面失效与频繁采样之间的双重风险;

(3)在 GCC (GNU Compiler Collection) 编译器与 OpenTuner 框架上基于 PolyBench 测试集进行实验,验证所提方法的性能收益与鲁棒性。

1 相关工作

1.1 编译器自动调优技术

Autotuning 旨在面向特定程序、特定平台和特定输入数据,自动选择编译优化配置(如选项组合、参数阈值与优化序列等),从而得到超越基于经验分类的固定优化等级(如-O选项)所带来的性能提升。如图1所示,编译自动调优技术在编译选项所代表的优化遍的组合、顺序,以及参数阈值构成的高维空间中进行搜索,通过编译、运行获得性能反馈指标,并将其提供给 Autotuning 搜索算法,搜索算法根据输入更新其内部状态并生成下次迭代使用的编译配置,从而完成一次迭代更新。通过循环迭代, Autotuning 框架保存搜索历史并记录性能指标最优的编译配置,获得最终调优结果。迭代编译自动搜索框架可以寻找适用于特定程序与平台的优化配置,以弥补固定优化等级与静态启发式优化不够灵活造成的局限性^[6]。

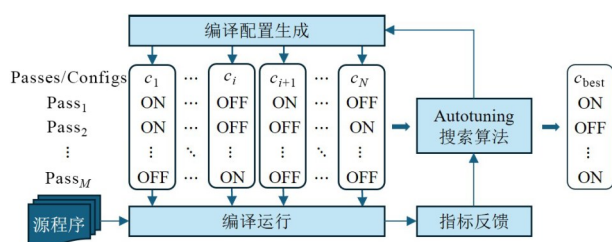


图1 Autotuning 自动搜索框架

Figure 1 Automatic search framework for autotuning

早期研究确立了迭代编译自动调优的基本范式, Bodin 等人将该问题视为非线性优化空间中的迭代搜索问题,指出其成本高但能获得显著收益^[1]。优化选择和序列重排问题随后被确立为核心难题。Kulkarni 等用进化搜索算法寻找更有效的优化序列,并强调迭代搜索的评估代价较高,需配合剪枝等技术以降低搜索开销^[3]。Triantafyllis 等进一步设计 OSE (Optimization-Space Exploration) 编译器,由启发式生成少量候选配置并以实测择优,使迭代编译成本可控^[2]。COLE (Compiler Optimization Level Exploration) 框架进一步将调优扩展为多目标优化,以运行时间与编译时间等指标进行折中选择^[13]。Chen 等人在 1 000 个数据集上评估迭代优化,揭示最优配置对输入高度敏感的问题^[14]。

随着编译优化选项和参数的增加,编译选项搜索空间扩大,使用机器学习工具加速优化过程以及建立

通用框架成为重要研究方向。相关研究主要集中在使用机器学习技术进行编译选项空间的搜索和决策优化,从而进一步降低搜索开销与提升优化效率。cTuning 及后续 Collective Mind/Collective Knowledge (CM/CK) 工作^[15-16]是迭代编译自动调优领域的重要基础。该类工作通过协同优化基础设施和公共优化知识库,将不同用户、程序、输入、平台和编译配置下的实验结果组织起来,以支持优化经验的共享、复用和预测。Agakov 等人以预测模型聚焦迭代搜索区域^[5], Milepost GCC 结合 CK 优化仓库,探索和学习程序特征与优化决策之间的映射关系,并进一步将其集成到通用编译器 GCC 中^[7]。

近年来,自动调优研究进一步尝试利用搜索空间中的结构信息对搜索空间进行构造与裁剪。例如, BaCO (Bayesian Compiler Optimization) 优化器使用贝叶斯优化处理现代编译器中离散选项、参数共存的复杂搜索空间^[17]; CFSCA (Critical-Flag Selection based Compiler Autotuning) 通过关键选项选择减少编译选项组合空间^[18]; CompTuner 使用少量实测数据预测不同优化组合性能^[19]; GroupTuner 进一步面向 GCC 选项组进行局部变异搜索^[20]; 近期工作还通过优化协同关系构建更紧凑的搜索空间。另外, MLGPerf 基于 MLGO (Machine Learning Guided Optimization) 将以缩小程序体积为目标的基于机器学习的内联策略扩展到以性能优化为目标的场景^[21], ACPO 则提供了更通用的机器学习编译器框架^[22]。这些工具将机器学习模型集成到 LLVM 编译器内部优化器中,用于内联、循环展开、寄存器分配等决策。Compiler-R1^[23]和 Large Language Model Compiler^[24]的出现表明,大语言模型和强化学习也开始用于编译器自动调优、程序中

间表示建模、代码性能与尺寸优化等任务。在迭代自动调优工具框架的发展方面, Active Harmony 推动自动调优工具服务化^[25]。Tiwari 等人实现对编译参数的并行探索^[26]。OpenTuner 提供了可扩展的多目标自动调优框架,其关键设计是搜索技术集成,多种搜索器共享结果并采用多臂赌博机算法动态分配评测预算^[6]。

与上述工作不同,本文并不以构建优化知识库或改进编译器优化选择为主要目标,而是关注反馈优化场景下自动调优与运行时剖面信息之间的耦合关系。迭代编译自动调优通常将编译器视为黑盒,以执行时间、代码大小、编译时间等外部指标作为搜索反馈;而在反馈优化融合场景中,性能提升还取决于剖面信息的编译配置及其与当前候选优化配置的一致性。当配置变化显著改变调用图、控制流或循环结构时,旧剖面可能出现映射失效或运行时信息代表性下降

的问题。

1.2 反馈导向优化技术

反馈导向优化依赖运行时剖面信息校正编译过程中优化器使用的静态启发式决策,在分支权重、内联与代码布局等优化决策上起到关键作用。按运行时剖面信息采集方式,程序追踪技术通常分为插桩与采样两类。

基于软件插桩的追踪技术在程序中插入计数器,从而精确获得基本块或控制流的边频率信息并供给编译器进行优化决策。Gprof(GNU profiler)以调用频率刻画程序热点,形成了经典的程序剖面特征的获取方式^[27]。Pettis 和 Hansen 的工作是通过记录执行频率识别热点代码,并据此改进代码布局以提升指令缓存效率^[4]。为进一步降低插桩成本并提升信息粒度,Ball 与 Larus 提出高效的路径剖析技术,以选择性插桩统计路径级频率,并在收集程序路径剖面后通过推理复原程序实际运行路径分布,使优化器可以聚焦于热路径上的优化^[28]。总体而言,基于插桩的程序剖面信息采集具有精度高、与程序控制流图直接对齐的优点,但需要执行额外的指令,可能会带来显著的性能开销。

基于采样的追踪技术以统计抽样和硬件计数器替代全量软件计数。Ammons 等人首先提出将硬件事件与控制流、调用上下文关联,从而得到对编译优化更有指向性的反馈信息^[29]。随后,SamplePGO 将剖面信息的生成、转换与注入纳入 LLVM 工作流,降低了 FDO 使用门槛^[30]。Wicht 等人利用现代处理器中提供的硬件信息采样重建边频率,开销仅为程序正常运行的 1%^[31];AutoFDO 编译优化器以及 BOLT 等后链接优化器也利用剖面信息面向数据中心应用实现函数和基本块的重排优化^[12,32-33]。总体上,基于采样的程序追踪可以显著降低全量追踪带来的开销,但是会引入统计噪声与误差,需要在采样数据处理阶段进行校准与跨版本可迁移性之间的权衡。主流编译器已经提供了对运行时剖面信息的收集与实施反馈优化的工程化机制。如图 2 所示,以 GCC 为例,通过 `-fprofile-generate` 选项生成运行时信息,再通过 `-fprofile-use` 选项利用运行时剖面信息完成 FDO 优化。

此外,FDO 的有效性高度依赖插桩或采样时所采用的编译选项配置。传统实践通常在 `-O2` 或 `-O3` 优化等级上应用 FDO,但这些选项组启动的优化选项并不一定是针对目标程序的最优配置。Milepost GCC 的研究表明,在不同程序上最优的配置存在显著差异^[7]。此外,FDO 相关的子选项同样需要精细调整,不当的选择可能引入负优化。Chen 等人通过实验证明,FDO 并非总能带来性能提升,在某些程序中甚至导致性能

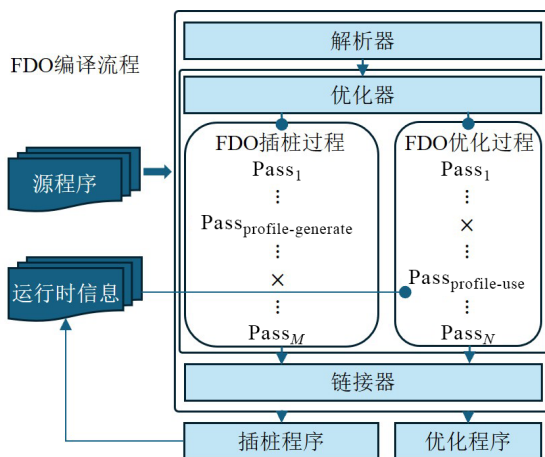


图2 FDO插桩与优化过程

Figure 2 FDO instrumentation and optimization process

回退^[34],这进一步说明了 FDO 选项调优的必要性。

综上所述,cTuning、Milepost GCC、OpenTuner 等工作已经系统证明了迭代编译和自动调优在编译选项空间中的有效性;FDO 相关工作则表明,运行时剖面信息可以改善内联、布局、分支预测等关键优化决策。然而,二者的直接融合会引入新的问题,即剖面信息本身由某一编译配置采集得到,其有效性依赖于该固有配置与当前优化配置之间的结构一致性。已有自动调优工作通常不使用运行时信息,也不决定何时更新运行时信息;已有 FDO 工作通常假设编译选项固定不变,更关注运行时信息如何采集、映射和修复。本文工作建立在 GCC 编译器已有的标准 FDO 工作流之上,研究将 FDO 优化流程纳入自动调优的迭代搜索过程中。

2 自动调优与反馈导向优化融合模式

传统自动调优搜索框架通常将编译优化器视为黑盒,在由优化选项及其参数构成的配置空间中,搜索器迭代生成候选编译选项配置 c ,并通过编译和运行获取性能指标 m ,以此不断更新搜索状态与策略。通过记录搜索过程中出现的最佳性能指标所对应的编译配置,返回针对目标程序的最终最优配置。以 OpenTuner 为代表的自动调优框架进一步将多种搜索算法进行集成组合,并引入多臂赌博机算法作为选择机制,在不同搜索策略之间进行自适应的概率选择,从而提升在复杂搜索空间中的探索效率与鲁棒性。

FDO 则显式引入运行时剖面信息 p 来指导启发式优化,如内联、代码布局、分支预测等关键决策的优化。为降低传统插桩反馈导向优化的两次构建和运行时软件采样带来的额外开销,SamplePGO、AutoFDO 等对处理器性能监控单元产生的硬件事件

进行采样,从而还原出近似频次和边剖面信息,使运行时剖面采集更适合在生产环境持续部署^[12,30];与此同时,采样剖面存在噪声与缺失,需要在剖面信息到源程序的映射过程中进行平滑处理才能稳定驱动 FDO 获得性能提升^[35]。

2.1 问题建模

自动调优工具擅长在静态配置空间中搜索近似最优解,但其用于更新搜索状态的反馈信息通常为单一性能标量,且未利用程序的动态运行时行为特征。此外 FDO 相关选项本身也可构成可调配置空间,如表 1 所示。因此, FDO 与 Autotuning 融合还面临采样时使用何种编译配置的选择性挑战。将自动调优框架与 FDO 结合,既能将动态运行时剖面数据用于自动调优框架的搜索过程,又能利用框架的探索能力寻找近似最佳的 FDO 配置。

表 1 FDO 优化选项及其对应优化等级

Table 1 FDO optimization options and their corresponding optimization levels

反馈导向优化选项	优化等级
-fbranch-probabilities	—
-fprofile-values	—
-funroll-loops	03
-fpeel-loops	—
-ftracer	—
-fvpt	02
-finline-functions	02
-fipa-cp	03
-fipa-cp-clone	02
-fipa-bit-cp	03
-fpredictive-commoning	03
-fsplit-loops	03
-funswitch-loops	03
-fgcse-after-reload	02
-ftree-loop-vectorize	02
-ftree-slp-vectorize	03
-fvect-cost-model=dynamic	—
-ftree-loop-distribute-patterns	—
-fprofile-reorder-functions	—

然而,两者结合也面临新的技术挑战,即如何决定剖面信息的更新时机和频率。一方面,频繁采样会显著增加调优开销并降低迭代搜索速度;另一方面,复用同一份剖面数据可能面临剖面数据失效或与当前优化配置不匹配的问题。因为剖面是基于某一构建版本与优化配置得到的,当配置改变导致控制流和调用关系发生变化时,旧的运行时剖面信息可能无法准确反映或不再具有程序热点部分运行时特点代表性。

为形式化描述,用式(1)表示一次评估:

$$y(c; c^{\text{prof}}) = \text{Eval}(\text{Build}(c, \text{Profile}(c^{\text{prof}}))) \quad (1)$$

其中, $\text{Profile}(c^{\text{prof}})$ 表示在构建配置 c^{prof} 下编译运行收集到的剖面数据 P , $\text{Build}(c, P)$ 表示用优化配置 c 和剖面数据 P 生成的二进制程序, $\text{Eval}(\cdot)$ 为运行测量得到的性能指标。Autotuning 和 FDO 融合的关键在于在有限预算 B 下选择一条由 $\{c_i, c_i^{\text{prof}}\}$ 构成的高效搜索轨迹,保证迭代搜索开销之和 $\sum_i \text{Cost}(c_i, c_i^{\text{prof}})$ 小于预算 B , 且最终使性能指标 y 最优。 $\text{Cost}(\cdot)$ 同时包含编译、采样与评测运行的时间成本。

本文对自动调优框架与 FDO 的融合方案进行了形式化统一,如算法 1 所示。具体而言,由自动调优框架产生候选配置 c_i ,并在迭代搜索中维护当前运行时剖面信息 P 与其对应的编译配置 c^{prof} ,并由采样更新策略决定何时更新采样信息。算法 1 给出了本文提出的自动调优框架与 FDO 融合算法。其中,搜索器 S 接收反馈指标并产生下一轮迭代使用的编译配置,构成 Autotuning 的迭代循环。算法在迭代搜索过程中维护当前剖面 P 及其优化配置 c^{prof} ,每轮先由搜索器 S 生成候选配置 c_i ,再由更新策略 $U(\cdot)$ 判断是否需要以当前配置 c_i 重新采样;若不触发,则继续复用已有剖面。随后利用当前剖面完成 FDO 构建与性能评测,得到反馈指标 y_i ,并据此更新搜索器状态。迭代结束后输出历史最优配置及对应二进制程序。由此,剖面数据采集、剖面数据复用与优化配置搜索被统一到同一优化循环中,不同融合模式体现在运行时信息更新策略 $U(\cdot)$ 的具体设计。本节后续将具体介绍基于两阶段搜索、固定间隔采样以及条件采样三种更新策略的设计。

算法 1 Autotuning 与 FDO 融合算法

输入: 搜索器 S , 初始配置 c_0 , 剖面更新策略 $U(\cdot)$

输出: $c_{\text{best}}, \text{Binary}_{\text{best}}$

初始化状态: 运行时剖面信息 $P \leftarrow \emptyset$, 运行时剖面信息来源配置

1. $c^{\text{prof}} \leftarrow \emptyset$, 历史最佳配置 $c_{\text{best}} \leftarrow \emptyset$

2. For $t = 1, 2, \dots, T$ do

3. $c_t \leftarrow S.\text{generate}()$

4. If $U.\text{needProfile}(t, c_t, c^{\text{prof}})$ then

5. $c^{\text{prof}} \leftarrow c_t, P \leftarrow \text{Profile}(c^{\text{prof}})$

6. End if

7. $y_t \leftarrow \text{Eval}(\text{Build}(c_t, P))$

8. $S.\text{update}(c_t, y_t)$

9. End for

2.2 融合模式

2.2.1 基于两阶段搜索的融合模式

FDO 技术基于编译器优化器实现,因此其相关选

项本身部分存在于自动调优框架的搜索空间内(如表1所示)。如图3所示,为了使FDO技术使用统一的运行时剖面数据,可以将编译选项空间划分为FDO相关选项和FDO无关选项。首先用自动调优框架在FDO无关选项构成的搜索空间找出较优基础配置,进行采样获得剖面数据,再使用自动调优框架对FDO

相关选项完成二次搜索。即形式化地将配置空间拆分为两部分:

$$c = \{c^{\text{Non-FDO}}, c^{\text{FDO}}\} \quad (2)$$

其中, $c^{\text{Non-FDO}}$ 为FDO无关选项, c^{FDO} 为FDO相关选项,例如利用剖面信息驱动内联、布局、值利用等策略。两阶段搜索目标可由下列公式表示。

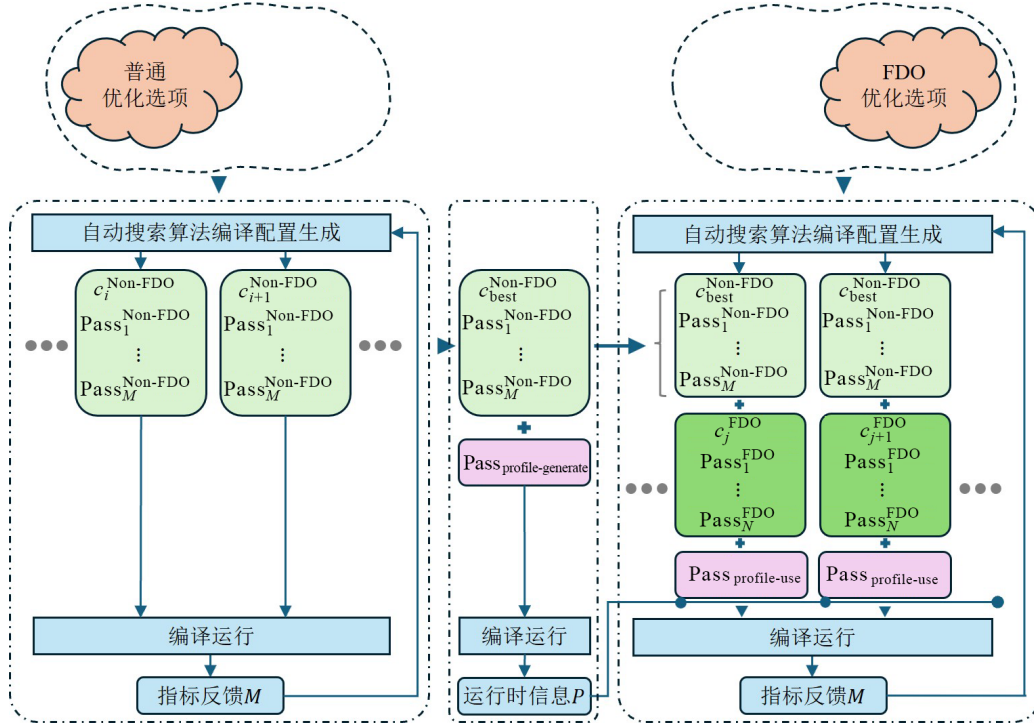


图3 基于两阶段搜索的融合模式

Figure 3 Integration mode based on two-stage search

第一阶段,固定 $c^{\text{FDO}} = \{\text{OFF}\}$, 搜索:

$$c^* = \underset{c^{\text{Non-FDO}}}{\operatorname{argmin}} \operatorname{Eval} \left(\operatorname{Build} \left(c^{\text{Non-FDO}}, c_{\text{OFF}}^{\text{FDO}}, \emptyset \right) \right) \quad (3)$$

在第二阶段,首先以 c^* 为固定编译配置,进行采样获得 $p^* = \operatorname{Profile} (c^*)$,随后在 c_{FDO} 上搜索:

$$c_{\text{FDO}}^* = \underset{c^{\text{FDO}}}{\operatorname{argmin}} \operatorname{Eval} \left(\operatorname{Build} \left(c^*, c^{\text{FDO}}, p^* \right) \right) \quad (4)$$

该方式仅采样一次,且由于第二阶段固定非FDO选项构成的配置 c^* ,采样获得的剖面信息与程序结构的变化不会产生显著偏差,可以使搜索过程更加稳定。但是两阶段分离搜索会弱化选项交互,即某些非FDO选项 $c^{\text{Non-FDO}}$ 的变化会显著改变热点结构(例如改变调用图或循环结构),则在第二阶段使用 c^* 编译配置运行采集到的运行时剖面信息可能无法代表其他潜在更优区域的实际运行情况。此外,若第二阶段允许 c^{FDO} 涵盖显著改变结构的策略(如激进内联或循环变换相关阈值),仍可能导致运行时剖面信息逐步偏离实际运行情况。针对程序变化导致的运行时偏

差问题,已有工作在二进制层面系统研究并提出匹配策略^[35],说明在结构变化下复用运行时剖面信息本身是一个需要显式处理的挑战问题。

2.2.2 基于固定间隔采样的融合模式

为了克服程序编译配置变化导致的运行时剖面信息变化的问题,可以在迭代搜索生成每一次编译配置更新时进行一次重新采样,使用得到的运行时剖面信息进行优化,并将优化后的结果作为性能反馈更新迭代搜索的算法状态,如图4所示。迭代搜索的每一步进行 $c^{\text{prof}} = c_i, p = \operatorname{Profile} (c_i), y_i = \operatorname{Eval} \left(\operatorname{Build} (c_i, p) \right)$ 对运行时剖面信息进行更新。为了减少频繁的采样开销,定义固定周期 K , 在每 K 次迭代搜索后进行重新采样。该方式实现简单、采样开销可控,能周期性纠偏剖面信息,但是固定的采样周期缺乏灵活性,当搜索早期结构变化频繁时 K 过大会导致剖面信息迅速失效;当后期收敛稳定时 K 过小则会造成冗余开销。从统一触发策略的角度看,该方法属于配置无关的时间触发采样,其触发条件仅依赖迭代计数而不依

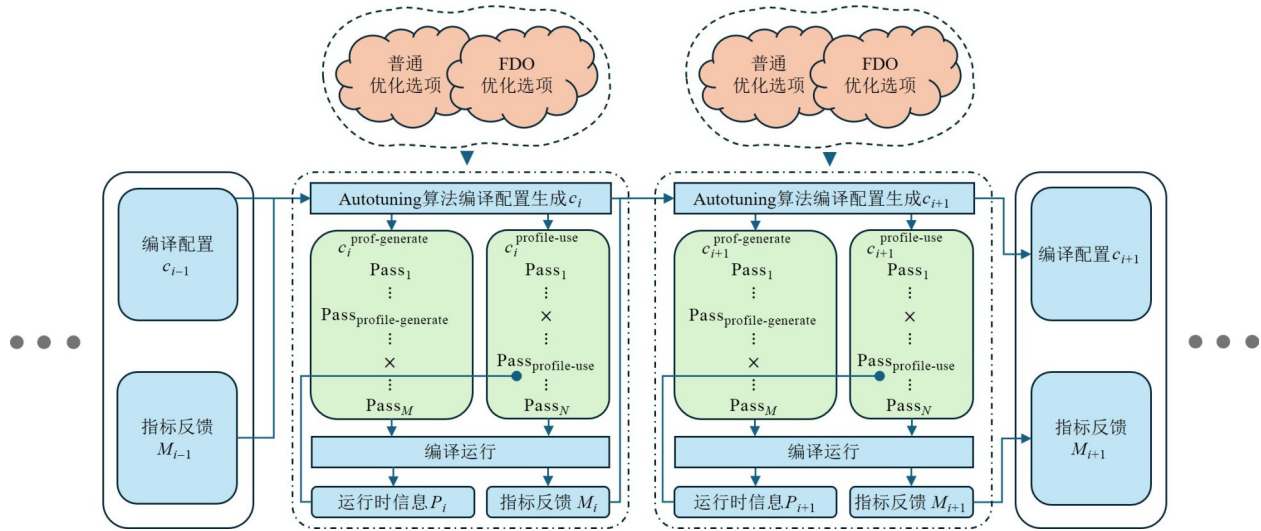


图4 基于固定间隔采样的融合模式

Figure 4 Integration mode based on fixed-interval sampling

赖当前候选配置与剖面来源配置之间的偏离程度。当 $K=1$ 时,该方法可以最大程度避免运行时剖面信息失效问题。但额外带来的采样和处理开销会显著降低单位时间内可探索的优化配置数量,同时采样误差会叠加到搜索反馈中放大噪声,从而导致收敛变慢。

2.2.3 基于条件采样的融合模式

在统一剖面更新策略视角下,两阶段分离搜索与固定间隔采样搜索可以看作触发条件仅依赖时间/迭

代计数的特例。为了解决上述问题,本文提出配置感知的条件采样策略,如图5所示,即在连续的迭代搜索过程中允许有条件地更新运行时剖面信息。根据自动调优技术的搜索逻辑,编译配置通常呈现局部变化,因此可以采用设定少数采样点的形式进行有条件采样,即仅当搜索到符合预设的采样条件时再进行采样。本文将可调选项按破坏程序结构(从而导致运行时剖面信息失效)的风险划分为三组:

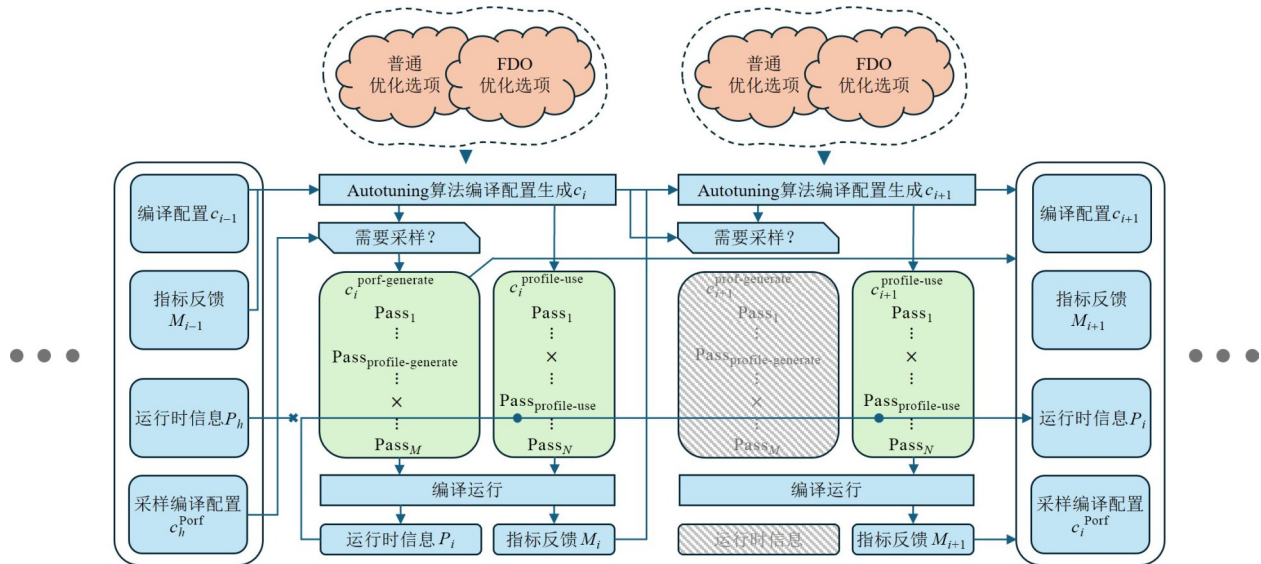


图5 基于条件采样的融合模式

Figure 5 Integration mode based on conditional sampling

$\mathcal{O}_H$: 强可能性破坏结构选项。这类选项通常会改变调用图、程序控制流图或循环和向量化结构,导致采样样本到基本块与边的映射与热点分布发生根

本变化。由于FDO优化依赖将剖面信息映射回程序位置,并进一步推断与平滑得到边频等结构化信息。当结构被显著改写,运行时剖面信息更易出现覆盖缺

失、错配与误导。

$\mathcal{O}_L$:弱可能性破坏结构选项。这类选项通常只在某些特定控制流及数据流下改变代码结构或热点分布,风险较低但不可忽略,因此采用累积偏离阈值来决定是否重采样。

$\mathcal{O}_N$:无关/低相关选项。这类选项主要影响局部代码生成或不改变剖面映射所需的结构锚点,通常无需因其变化进行重新采样。

本文依据 GCC 优化选项及其说明进行三类划分,如表 2 所示。其中 $\mathcal{O}_H$ (高风险) 包含 17 个典型结构性优化 (如 `-finline-functions`、`-floop-unroll-and-jam` 等), $\mathcal{O}_L$ (弱风险) 包含 6 个可能引发部分细小局部结构变化的优化选项 (如 `-fgcse-after-reload`、`-fpredictive-commoning` 等), 其余 95 个为 $\mathcal{O}_N$, 为对程序结构不存在重大改变的选项。设当前使用的运行时剖面信息的来源配置为 c^{prof} , 当前候选配置为 c , 定义下述两类偏离检测。

强风险触发: 若存在任意 $o \in \mathcal{O}_H$ 使得 $c_i(o) \neq c^{\text{prof}}(o)$ 则立即触发重新采样 $c^{\text{prof}} \leftarrow c_i$,

表 2 选项风险等级

Table 2 Risk levels of optimization options

风险等级	选项
高风险	<code>-finline-functions</code>
	<code>-fipa-bit-cp</code>
	<code>-fipa-cp</code>
	<code>-fipa-cp-clone</code>
	<code>-floop-interchange</code>
	<code>-floop-unroll-and-jam</code>
	<code>-fpeel-loops</code>
	<code>-fsplit-loops</code>
	<code>-ftracer</code>
	<code>-ftree-loop-distribute-patterns</code>
	<code>-ftree-loop-distribution</code>
	<code>-ftree-loop-vectorize</code>
	<code>-ftree-slp-vectorize</code>
	<code>-funroll-loops</code>
	<code>-funswitch-loops</code>
	<code>-fvect-cost-model=dynamic</code>
	<code>-fversion-loops-for-strides</code>
低风险	<code>-fbranch-probabilities</code>
	<code>-fgcse-after-reload</code>
	<code>-fpredictive-commoning</code>
	<code>-fprofile-reorder-functions</code>
	<code>-fprofile-values</code>
无风险	<code>-fvpt</code>
	<code>-fauto-inc-dec</code>
	

$p \leftarrow \text{Profile}(c^{\text{prof}})$ 。

弱风险触发: 仅当强风险选项未触发重新采样机制时, 计算弱风险组选项上的汉明距离 (Hamming distance):

$$d_L(c_i, c^{\text{prof}}) = \sum_{o \in \mathcal{O}_L} 1[c_i(o) \neq c^{\text{prof}}(o)] \quad (5)$$

当 $d_L(c_i, c^{\text{prof}}) \geq \tau$ 时触发重新采样, 否则沿用之前的运行时剖面信息。该机制用于区分低风险选项的单次局部变化与多项变化累积后可能造成的结构偏离, 若单次 Autotuning 选择探索的编译配置只改变部分低风险选项和无风险选项, 则该变化无需更新运行时剖面信息。但当搜索算法生成的编译配置修改涉及高风险选项或涉及多个低风险选项, 且与目前使用的剖面信息对应的编译配置偏离累积到一定程度时, 需更新运行时剖面信息以避免其失效, 从而避免 FDO 使用失效信息带来性能下降。

3 实验结果与分析

3.1 实验环境与测试用例

本文在配备飞腾处理器的服务器上对 Autotuning 与 FDO 的融合方案进行性能评估。该机器的 CPU 型号为 Phytium2000+/64, 配备 64 GB 内存, 8 个 NUMA 节点, 主频 2.0 GHz, 共有 4 MiB 一级缓存, 32 MiB 二级缓存, 无共享三级缓存。为了降低系统抖动造成的影响, 实验中统一设置 CPU 固定频率、核心绑定和 NUMA 亲和性。软件方面, 该机器使用 Kylin Advance SP3 国产操作系统, 内核版本为 4.19.90。编译器统一采用 GCC 12.2.0。为启用 FDO, 首次编译时使用 `-fprofile-generate` 选项采集运行时剖面信息, 二次编译时使用 `-fprofile-use` 读取运行时剖面信息实施 FDO 优化。使用 OpenTuner 作为自动调优框架。

本文选用在编译器迭代搜索领域广泛使用的测试集 PolyBench 进行测试。如表 3 所示, 该测试集提供统一的内核计时与监控支持, 其测试用例覆盖线性代数、Stencil 等多个典型计算内核, 其测试结果具有较强代表性。

3.2 对比方法和评价指标

为保证公平性, 实验对每个测试均设置统一的 30 min 的调优时间 (包含编译、运行以及运行时剖面信息的采集与反馈导向优化再编译的开销)。基于此, 本文使用基于 -O3 选项进行 FDO 优化的运行时间作为对比基准, 与 OpenTuner 以及本文所提出的三种融合方案的运行时间进行比较。需要说明的是, ML-GOPerf、ACPO、Compiler-R1 和 Large Language Model Compiler 等近期工作主要基于 LLVM 项目及其衍生工具, 研究对象包括序列搜索与优化; 而本文实验基于 GCC 的 `-fprofile-generate`/`-fprofile-use` 工作流, 核心变量

表3 PolyBench 测试程序
Table 3 PolyBench benchmark programs

分类	程序名称
统计分析	correlation, covariance
BLAS 类算子	gemm, symm, syr, syr2k, trmm, gesummv, gemver
矩阵-向量/转置运算	atax, bicg, mvt
矩阵乘	2 mm, 3 mm, doitgen
矩阵分解与线性方程组求解	cholesky, lu, ludcmp, gramschmidt, trisolv, durbin
动态规划与图算法	dynprog, floyd-warshall
图像处理	reg_detect
迭代 stencil/PDE 求解	adi, fdt-d-2d, fdt-d-apml, jacobi-1d-imper, jacobi-2d-imper, seidel-2d

为剖面来源配置、剖面更新策略与 GCC 编译选项搜索反馈过程。由于两类工作的编译器基础设施、中间表示与优化管线、优化目标和反馈形式均不同,本文主要将其作为相关技术脉络进行讨论,而不进行直接数值对比。

对于默认 OpenTuner 搜索设置,实验在编译优化等级-O1/2/3 涉及的所有优化选项空间构成的普通选项搜索空间中进行黑盒搜索,评价指标为程序实际的运行时间。该方法不对运行时剖面信息进行采集,亦不使用 FDO 优化,代表自动调优框架在搜索预算内所能达到的优化能力。

在基于两阶段搜索的融合方案中,实验在第一阶段对编译组选项-O1/2/3 涉及的所有优化选项空间构成的搜索空间上进行 24 min 的搜索,获得在该空间上的最优选项配置,并使用该选项配置作为 FDO 中运行时采样的基础编译配置进行编译和采样,获得运行时剖面信息。随后,实验在第二阶段进行 6 min 的基于运行时剖面信息的搜索,搜索空间由 FDO 相关选项构成。在基于固定间隔采样的融合方案中,实验设置 $K=1$,即在每次迭代中进行一次采样,并根据程序实际运行得到的剖面信息进行反馈导向优化的二次编译。在基于条件采样的融合方案中,实验在普通选项空间与 FDO 选项空间构成的搜索空间中连续搜索 30 min,但在每次迭代过程中首先判断是否需要重新采样,若判断为需要则触发重新采样并更新运行时的剖面信息,否则复用已有的剖面信息。

对于测试集中的每个测试程序 b 与编译配置 c ,在该编译配置下编译运行程序 b 的运行时间为 $T(b, c)$,本文以最小化执行时间为调优目标。为跨程序比较结果,定义相对于使用默认-O3 的编译配置 c_{-03} 运行时间的加速比为

$$S(b, c) = \frac{T(b, c_{-03})}{T(b, c)} \quad (6)$$

跨测试集 B 的总体加速比用几何平均值给出,以避免极端值对总体的影响:

$$S_{\text{geo}}(c) = \exp \left(\frac{1}{|B|} \sum_{b \in B} \ln S(b, c) \right) \quad (7)$$

为进一步比较不同方法的收敛速度,本文记录搜索过程中已发现的最优加速比 $S_{\text{best}}(T)$,并定义两个时间指标: t_{95} 表示在调优过程中首次达到最终最优性能 95% 的时间,即满足 $S(t) \geq 0.95 S_{\text{best}}(T)$ 的最早时间; t_{best} 表示首次达到最终最优配置性能的时间,即满足 $S(t) = S_{\text{best}}(T)$ 的最早时间。本文在 30 个测试程序上计算并给出两项指标的中位数。各程序加速比结果如图 6 所示。

3.3 运行时间对比

结果如表 4 所示,其中下划线数据表示最优结果。在-O3 基础上使用默认 FDO 的几何平均加速比仅为 1.057 $\times$,且在 30 个测试程序中仅 15 个获得加速。这说明在 PolyBench 这类以规则循环与数值计算为主的工作负载中,FDO 并不一定能实现性能加速,其性能收益高度依赖于具体程序的控制流特征、运行时剖面信息质量。相比之下,OpenTuner 框架能在 26 个程序上优于-O3 基线性能,几何平均加速比为 1.091 $\times$,表明在测试集上探索优化选项组合本身就能带来一定的性能收益。FDO 与 Autotuning 融合的方法能够进一步实现性能加速(1.142 $\times$ 、1.165 $\times$ 、1.184 $\times$),表明其优势并非简单叠加,而是通过更好地运行时剖面信息使用时机与优化选项的选择,充分挖掘 FDO 和 Autotuning 工具的优化潜力。从带来超过基础-O3 性能的程序数量上来看,条件采样搜索技术并非数量上最多的方法,然而几何平均结果却显示条件采样搜索技术在若干程序上获得了更大幅度的提升,同时显著降低了严重负优化的风险。

以相对基准指标回退超 5% (即 $S < 0.95$) 作为粗粒度风险评价指标,FDO 在 4 个测试程序上造成性能下降; OpenTuner 在 1 个测试程序上性能下降; 两阶段搜索造成 3 个程序性能下降超 5%, 其中 2 个程序超 10%; 有 4 个程序 (其中 1 个超过 10%) 在使用间隔采样搜索后程序性能下降; 而条件采样搜索不仅提升了

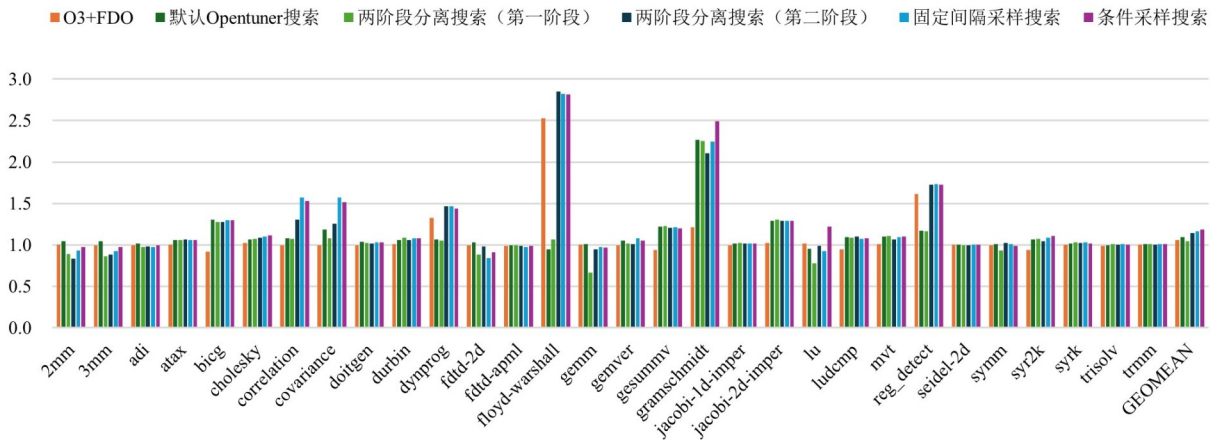


图6 三种融合模式在PolyBench测试集中各个测例上相对于-O3的加速比

Figure 6 Speedups of the three integration modes over -O3 on each PolyBench benchmark

表4 PolyBench整体测试结果

Table 4 Overall results on PolyBench

对比方法	几何平均加速比	迭代搜索次数	t_{95}/min	$t_{\text{best}}/\text{min}$
FDO	1.057×	—	—	—
OpenTuner	1.091×	379.6	0.57	14.01
两阶段搜索(第一阶段)	1.047×	260.9	—	—
两阶段搜索(最终)	1.142×	339.6	2.04	19.53
固定间隔采样	1.165×	304.5	1.96	18.81
条件采样	1.184×	307.0	0.56	10.99

平均性能,还显著缓解了引入运行时剖面信息带来的优化不稳定问题。这与前文对采样噪声以及运行时剖面信息与配置耦合的风险分析是一致的,即当运行时剖面信息更新机制能够在必要时重新采样,并在无需更新时复用已有剖面信息的原则下,系统更容易避免失效运行时剖面信息的误导。

3.4 收敛速度对比

融合方法引入运行时剖面信息的采集与二次编译会增加单次评测成本,从直觉上可能降低搜索吞吐量并拖慢收敛速度。本节对这一权衡进行量化分析。在30 min搜索预算内,各方法的平均有效评测次数如表4所示。与默认OpenTuner搜索相比,固定间隔采样搜索与条件采样搜索的评测次数减少19%,这反映出运行时剖面信息的采集以及FDO二次编译带来明显额外开销。然而,最终性能却有明显提升,说明运行时剖面信息所带来的信息增益足以抵消搜索覆盖率下降可能造成的搜索结果劣化。

从收敛过程来看,设 $S_{\text{best}}(t)$ 为自动调优进行到时间 t 时已发现的最优加速比,根据3.2节定义的两类时间指标 t_{95} 与 t_{best} ,其中 t_{95} 代表达到最终最优配置性能95%的最短时间; t_{best} 代表达到最终最优配置的最短时间。在30个测试程序上取中位数得到各项测

试方法的两项指标如表4所示。该数据表明条件采样搜索不仅最优配置性能最高,而且在中位数意义下可以更快搜索到最优配置。这意味着条件采样并未因运行时剖面信息采样带来的开销而显著拖慢收敛,反而通过更高质量的反馈信号加速了搜索迭代的收敛。与之对比,两阶段搜索与固定间隔搜索的 t_{best} 均约为19 min。这与它们受到更严重的噪声影响、开销更加明显的剖面生成以及使用方式有关。对于 t_{95} ,默认OpenTuner搜索与条件采样搜索非常接近,说明即便将编译器视作黑盒,缺少运行时剖面信息,OpenTuner也可以迅速通过启发式探索算法搜索到较优配置。然而其最终最优配置显著低于条件采样搜索的结果,表明默认OpenTuner更难在剩余的搜索时间里继续逼近更高的性能上界。即仅使用单一指标反馈在高维组合空间迭代搜索后期往往进行大量无意义试错,而FDO为某些结构性决策提供了更强的导向,使搜索后期仍能持续获得可观的性能增益。

3.5 不同融合方案对比

两阶段搜索方法最终几何平均加速比为1.142×,相对默认OpenTuner提升4.66%,证明先在普通选项空间中搜索基础编译配置,再在反馈导向优化选项空间中进行进一步搜索的方法是行之有效的。尤其是

从阶段内部看,第一阶段的平均加速比仅为 1.047 $\times$,而第二阶段引入运行时剖面信息并在 FDO 选项子空间二次搜索后提升至 1.142 $\times$,即相对第一阶段提升约 9.13%,说明第二阶段对性能提升贡献显著。这些结果充分支撑了本文的核心动机,即对优化相关选项集合与运行时剖面信息的使用策略进行细粒度研究是值得的。

然而,两阶段搜索技术明显落后于条件采样搜索与固定间隔搜索,其局限主要体现在两点:首先,在使用两阶段搜索时普通选项与 FDO 选项之间的交互被顺序化处理。第一阶段的搜索目标是无运行时剖面信息条件下的最优普通选项配置,而第二阶段的目标是固定运行时剖面信息及普通优化选项配置下的最优反馈导向优化选项配置。在普通优化选项与 FDO 选项之间存在重叠的情况下,如果某些选项的最优策略强依赖于运行时剖面信息驱动(例如与内联/布局的协同),顺序化会使搜索更难触及真正的联合最优配置;其次,剖面信息复用会带来失效风险。第二阶段固定使用从第一阶段最优配置中采集的剖面信息,尽管第二阶段只在 FDO 子空间内搜索,其中仍可能包含改变代码结构或热点分布的选项,导致运行时剖面信息与实际编译配置发生偏离。在最终结果中也观察到两阶段方法存在更大的性能劣化(如 2 mm、3 mm 程序性能劣化超过 10%),这与固定运行时剖面信息与编译配置改变导致的结构变化可能引发的误导搜索相符合。因此,两阶段搜索方法体现了典型的开销与实现难度的权衡,采样次数少、实现简单,但对交互关系刻画不足,且对运行时剖面信息是否如实对应程序结构更敏感。

固定间隔采样搜索等价于每次迭代都进行运行时剖面信息采样并据此进行 FDO 二次优化编译,其最终几何平均为 1.165 $\times$,显著优于两阶段搜索与默认 OpenTuner 搜索,并在 30 个测试程序中有 23 个超过 -O3 基准性能。在测试程序最优次数角度上,固定间隔采样搜索在四种搜索方法中成为最优的次数为 10 次,仅次于默认 OpenTuner 搜索的 11 次,说明该策略确实能在若干程序上提供强有力的性能提升。然而其整体加速性能略逊于条件采样(1.184 $\times$),并且在鲁棒性上存在更明显的风险:固定间隔采样搜索方法下有 4 个程序性能劣化多于 5%,最差回退出现在 fdtd-2d 程序(0.838 $\times$),并且在 lu 等程序上也出现较明显回退(0.925 $\times$)。结合方法分析,导致这类现象的原因可能有两点:首先,间隔采样导致评测成本提高进而致使探索覆盖率不足。固定间隔采样将运行时剖面信息采集成本增加到每个候选配置的迭代搜索上,使得 30 min 内评测次数下降到约 304 次(较默认下降约

20%)。在高维离散空间中,探索不足更容易陷入局部最优,或更频繁的搜索性能劣化的配置。其次,每次迭代搜索都进行重新采样意味着反馈值不仅取决于配置 c ,还取决于采样得到的运行时剖面信息的随机误差与代表性。在 FDO 中,样本缺失与噪声需要在二次优化编译时被平滑才能稳定提升性能。所以,当剖面信息在每次迭代都变化时,搜索器实际上面对的是一个非平稳的黑盒目标,可能导致收敛更慢,并更难可靠地区分编译配置变化带来的真实收益与采样误差带来的偶然波动。

条件采样搜索技术获得最优的综合结果,最终几何平均 1.184 $\times$,相对默认 OpenTuner 搜索提升 8.46%;在收敛速度方面,其最佳配置最短搜索时间在测试集上的中位数分钟为 $t_{\text{best}} = 10.99$,在四种方法中最优。在性能提升鲁棒性方面,只有 fdtd-2d 一个程序搜索结果性能劣化大于 5%,且不存在大于 10% 的结果。该结果与条件采样搜索的设计目标一致,即通过条件触发的运行时剖面更新机制,将采样资源集中用在真正可能导致运行时剖面信息失效的结构变化上,从而在其余迭代中复用已有运行时剖面信息,同时获得较高的剖面信息利用效率。

从机制层面解释,条件采样搜索之所以能优于固定间隔搜索,可能因为获得稳定的评测反馈有利于搜索算法做出可靠比较。当连续若干编译配置复用同一份运行时剖面信息,其随机误差对这些编译配置的影响是共享的,搜索器更容易基于相对比较做出稳定的决策;而当检测到高风险结构变化时再更新剖面信息,则可避免长期使用的运行时剖面信息失效。条件采样与固定间隔在本实验中的评测次数接近(307 次与 304 次),说明二者搜索次数的差异并不是决定性因素。条件采样仍能取得更优结果,表明其优势更多来自重新采集运行时剖面信息的决策带来的信息质量提升与误导风险降低。例如在 lu 程序上,固定间隔采样搜索最终表现为性能劣化(0.925 $\times$),而条件采样搜索实现明显性能提升(1.223 $\times$)。

综上,条件采样搜索体现了本文的关键价值,即在不把采样成本扩散到每一次评测的前提下,仍能有效抑制剖面信息失效带来的风险,从而在性能和采样开销之间取得更好的平衡,同时兼顾性能和成本。

3.6 典型程序对比分析

为进一步理解上述结论,本文对若干代表性程序进行详细分析,说明融合策略的收益来源与可能的局限性。

在 floyd-warshall 测试程序上, Autotuning 和 FDO 融合策略可以带来可观性能提升,但默认 OpenTuner 搜索会出现性能劣化(0.950 $\times$),而单独使用 FDO 加速

比可以达到 2.525 $\times$, 融合搜索方法可以进一步提升 2.8 $\times$ 以上。这表明该程序高度依赖 FDO (例如分支偏置、热路径相关决策以及激进的布局或内联优化等), 纯黑盒搜索不仅难以挖掘收益, 甚至可能因为误配某些优化选项而造成性能劣化。融合方法能够在该类程序上显著改善程序最终性能, 是本文方法的重要应用场景。

在 lu 测试程序上, 固定间隔采样搜索最终性能劣化达 0.925 $\times$, 而条件采样搜索性能提升可达 1.223 $\times$ 。在该例中, 条件采样通过复用运行时剖面信息形成更稳定的比较基准, 同时在关键结构变化点进行重新采样, 最终获得更优解。这与前述条件采样搜索更具有鲁棒性的整体统计结果一致。

在 correlation 与 covariance 程序上, 固定间隔采样搜索成为四种方法中的最优方法, 条件采样略低。这说明, 当程序对运行时剖面信息与候选编译配置是否严格一致更敏感时, 固定间隔仍可能取得更高性能上界, 而条件采样搜索并不能保证对每个程序都能达到最优性能。

在 2 mm/3 mm/gemm 等典型矩阵乘程序上, 默认 OpenTuner 均为四种方法中最优, 融合搜索方法均出现不同程度回退。这表明计算密集型且分支可预测性高的简单规则循环不能从 FDO 中显著受益, 与 FDO 的特点一致。在这种场景下, 引入运行时剖面信息与 FDO 相关优化选项反而可能与某些优化选项 (例如特定循环展开或向量化策略) 发生不利交互, 也提示后续仍需在搜索空间裁剪、结构风险判别与剖面质量控制上继续优化。

条件采样在 ftdtd-2d 程序上出现约 9% 的性能劣化, 而默认 OpenTuner 搜索在该程序上反而有小幅提升 (1.031 $\times$)。这一对比表明条件采样搜索虽然整体鲁棒性强, 但仍可能在某些 stencil 类程序上受到运行时剖面信息误导或配置交互影响。

3.7 参数敏感性分析

条件采样中, 弱风险组选项的重采样条件为 $d_L(c_i, c^{\text{prof}}) \geq \tau$ 。为说明 τ 的设定依据, 本文在 8 个代表性 PolyBench 程序上 (2 mm、atax、correlation、ftdd-2d、floyd-warshall、gemm、lu、reg_detect) 对 $\tau \in \{1, 2, 4, 6, >6\}$ 进行了小规模敏感性分析。测试程序的选取标准为: 根据表 3 中 PolyBench 测试集的 8 类测试程序, 每类挑选一个代表性测试程序加入小规模参数敏感性分析。

结果如表 5 所示, 其中 $\tau=1$ 表示将全部低风险选项按高风险选项对待, 而 $\tau>6$ 表示低风险偏离永不单独触发重采样, 表中下划线数据表示最优结果。实验仍采用 -fprofile-generate/-fprofile-use workflow 进行剖面信息采集以及 FDO 优化, 并统一设置 30 min 调优

表 5 条件采样参数敏感性结果

Table 5 Parameter sensitivity result of conditional sampling

τ 取值	几何平均加速比	平均重采样次数
1	1.258 $\times$	195
2	<u>1.330$\times$</u>	<u>163</u>
4	1.322 $\times$	217
6	1.321 $\times$	270
>6	1.321 $\times$	303

时间预算, 并以 -O3 编译设置下的平均时间作为归一化基准。结果表明, $\tau=1$ 时的几何平均加速比仅为 1.258 $\times$, 明显低于其他设置, 说明对低风险选项过度敏感会导致频繁重采样并压缩有效搜索预算; $\tau \in \{4, 6, >6\}$ 的最终几何平均加速比分别为 1.322 $\times$ 、1.321 $\times$ 以及 1.321 $\times$, 虽接近 $\tau=2$ 时的最优加速比 1.330 $\times$, 但运行时剖面的失效次数增至 217、270 和 303 次, 明显高于 $\tau=2$ 时的 163 次, 说明阈值过大将增加旧剖面失效风险。因此, 本文最终采用 $\tau=2$ 作为默认阈值, 以在最终性能和剖面稳定性之间取得更稳健的折中。

4 结束语

本文针对 Autotuning 与 FDO 如何有效结合的问题, 提出并验证了一套可实现、可对比、可扩展的融合框架与策略。自动调优的优势在于能够在高维离散优化空间中自动探索更优编译配置, 但其通常将编译器视为黑盒, 反馈信号多为单一性能指标, 难以直接利用运行时行为信息; FDO 能够提供热点与频率等动态信息, 提升内联、布局等关键决策的质量, 但其引入的运行时剖面信息采集及二次优化流程会带来额外开销, 并面临剖面失效与代表性等问题。为此, 本文以运行时剖面信息更新的时机与频率为两者融合的核心问题, 在 OpenTuner 提供的通用搜索框架上设计并实现了三类融合模式, 将搜索策略与运行时剖面信息的采集、复用和更新机制解耦, 使不同剖面更新策略能够在统一框架下进行组合与比较。

在实验层面, 本文在 PolyBench 基准测试集上进行了统一预算下的对比评测。结果表明本文的融合方法能够获得更高的最终性能, 其中条件采样策略取得最佳综合表现, 几何平均加速比达到 1.184 $\times$, 这表明, 合理管理运行时剖面更新策略是自动调优与反馈导向优化互补融合的关键。

致谢 本研究成果得到 CCF-飞腾基金资助。

参考文献

- [1] Bodin F, Kisuki T, Knijnenburg P, et al. Iterative compilation in a non-linear optimisation space[C]//Workshop on

- Profile and Feedback-Directed Compilation. Paris, France, 1998.
- [2] Triantafyllis S, Vachharajani M, Vachharajani N, et al. Compiler optimization-space exploration[C]//International Symposium on Code Generation and Optimization, 2003. CGO 2003. Piscataway: IEEE, 2003: 204-215.
- [3] Kulkarni P, Hines S, Hiser J, et al. Fast searches for effective optimization phase sequences[J]. ACM SIGPLAN Notices, 2004, 39(6): 171-182.
- [4] Pettis K, Hansen R C. Profile guided code positioning[C]//ACM SIGPLAN Notices. New York: ACM, 1990: 16-27.
- [5] Agakov F, Bonilla E, Cavazos J, et al. Using machine learning to focus iterative optimization[C]//International Symposium on Code Generation and Optimization. Piscataway: IEEE, 2006: 11pp.-11305.
- [6] Ansel J, Kamil S, Veeramachaneni K, et al. OpenTuner: An extensible framework for program autotuning[C]//Proceedings of the 23rd International Conference on Parallel Architectures and Compilation. New York: ACM, 2014: 303-316.
- [7] Fursin G, Kashnikov Y, Memon A W, et al. Milepost GCC: Machine learning enabled self-tuning compiler[J]. International Journal of Parallel Programming, 2011, 39(3): 296-327.
- [8] Smith M D. Overcoming the challenges to feedback-directed optimization (Keynote Talk)[J]. ACM SIGPLAN Notices, 2000, 35(7): 1-11.
- [9] Ramasamy V, Yuan P, Chen D, et al. Feedback-directed optimizations in GCC with estimated edge profiles from hardware event sampling[C]//Proceedings of GCC summit 2008. 2008: 87-102.
- [10] Kim H, Suleman M A, Mutlu O, et al. 2D-profiling: Detecting input-dependent branches with a single input data set[C]//International Symposium on Code Generation and Optimization. Piscataway: IEEE, 2006: 11pp.-11172.
- [11] Berube P, Amaral J N. Benchmark design for robust profile-directed optimization[C]//Proceedings of the 2007 SPEC Benchmark Workshop. Austin, TX, USA, 2007.
- [12] Chen D H, Li D X, Moseley T. AutoFDO: Automatic feedback-directed optimization for warehouse-scale applications[C]//Proceedings of the 2016 International Symposium on Code Generation and Optimization. New York: ACM, 2016: 12-23.
- [13] Hoste K, Eeckhout L. Cole: Compiler optimization level exploration[C]//Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization. New York: ACM, 2008: 165-174.
- [14] Chen Y, Huang Y J, Eeckhout L, et al. Evaluating iterative optimization across 1000 datasets[C]//Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2010: 448-459.
- [15] Fursin G. Collective Mind: Cleaning up the research and experimentation mess in computer engineering using crowdsourcing, big data and machine learning[PP/OL]. V1. arXiv (2013-08-11) [2026-04-01]. <https://arxiv.org/abs/1308.2410>.
- [16] Fursin G, Temam O. Collective optimization: A practical collaborative approach[J]. ACM Transactions on Architecture and Code Optimization, 2010, 7(4): 1-29.
- [17] Hellsten E O, Souza A, Lenfers J, et al. BaCO: A fast and portable Bayesian compiler optimization framework[C]//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4. New York: ACM, 2024: 19-42.
- [18] Zhu M X, Hao D. Compiler auto-tuning via critical flag selection[C]//2023 38th IEEE/ACM International Conference on Automated Software Engineering. Piscataway: IEEE, 2023: 1000-1011.
- [19] Zhu M X, Hao D, Chen J J. Compiler autotuning through multiple-phase learning[J]. ACM Transactions on Software Engineering and Methodology, 2024, 33(4): 1-38.
- [20] Gao B Y, Yao M Y, Wang Z M, et al. Grouptuner: Efficient group-aware compiler auto-tuning[C]//Proceedings of the 26th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems. New York: ACM, 2025: 122-133.
- [21] Ashouri A H, Elhoushi M, Hua Y Z, et al. Work-in-progress: MLGOPerf: An ML guided inliner to optimize performance[C]//2022 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems. Piscataway: IEEE, 2022: 3-4.
- [22] Ashouri A H, Manzoor M A, Vu M, et al. Work-in-progress: ACPO: An AI-enabled compiler framework[C]//2024 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems. Piscataway: IEEE, 2024: 21.
- [23] Pan H, Lin H, Luo H, et al. Compiler-R1: Towards agentic compiler auto-tuning with reinforcement learning[C]//The Thirty-Ninth Annual Conference on Neural Information Processing Systems. 2026.
- [24] Cummins C, Seeker V, Grubisic D, et al. LLM compiler: Foundation language models for compiler optimization[C]//Proceedings of the 34th ACM SIGPLAN International Conference on Compiler Construction. New York: ACM,

- 2025: 141-153.
- [25] Tapus C, Chung I H, Hollingsworth J K. Active harmony: Towards automated performance tuning[C]//SC'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing. Piscataway: IEEE, 2002: 44.
- [26] Tiwari A, Chen C, Chame J, et al. A scalable auto-tuning framework for compiler optimization[C]//2009 IEEE International Symposium on Parallel & Distributed Processing. Piscataway: IEEE, 2009: 1-12.
- [27] Graham S L, Kessler P B, McKusick M K. Gprof: A call graph execution profiler[C]//Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction. New York: ACM, 1982: 120-126.
- [28] Ball T, Larus J R. Efficient path profiling[C]//Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO 29. Piscataway: IEEE, 1996: 46-57.
- [29] Ammons G, Ball T, Larus J R. Exploiting hardware performance counters with flow and context sensitive profiling[C]//Proceedings of the ACM SIGPLAN 1997 Conference on Programming Language Design and Implementation. New York: ACM, 1997: 85-96.
- [30] Novillo D. SamplePGO - the power of profile guided optimizations without the usability burden[C]//2014 LLVM Compiler Infrastructure in HPC. Piscataway: IEEE, 2014: 22-28.
- [31] Wicht B, Vitillo R A, Chen D, et al. Hardware counted profile-guided optimization[PP/OL]. arXiv (2014-11-24) [2026-04-01]. <https://arxiv.org/abs/1411.6361>.
- [32] Panchenko M, Auler R, Nell B, et al. BOLT: A practical binary optimizer for data centers and beyond[C]//2019 IEEE/ACM International Symposium on Code Generation and Optimization. Piscataway: IEEE, 2019: 2-14.
- [33] Panchenko M, Auler R, Sakka L, et al. Lightning BOLT: Powerful, fast, and scalable binary optimization[C]//Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction. New York: ACM, 2021: 119-130.
- [34] Chen D H, Vachharajani N, Hundt R, et al. Taming hardware event samples for precise and versatile feedback directed optimizations[J]. IEEE Transactions on Computers, 2013, 62(2): 376-389.
- [35] Ayupov A, Panchenko M, Pupyrev S. Stale profile matching[C]//Proceedings of the 33rd ACM SIGPLAN International Conference on Compiler Construction. New York: ACM, 2024: 162-173.

作者简介



刘秉鑫 男, 1996年6月出生于河北省石家庄市。现为北京师范大学人工智能学院博士研究生。主要研究方向为编译优化。
E-mail: liubingxin@mail.bnu.edu.cn



石剑君 女, 1991年2月出生于河南省邓州市。现为北京师范大学人工智能学院博士后。主要研究方向为程序分析与优化、系统软件安全。
E-mail: shijianjun@bnu.edu.cn



黄颖晖 女, 2002年7月出生于广东省深圳市。现为北京师范大学人工智能学院硕士研究生。主要研究方向为自动调优。
E-mail: huangyinghui@mail.bnu.edu.cn



计卫星 男, 1980年2月出生于陕西省咸阳市。现为北京师范大学人工智能学院教授、博士生导师。主要研究方向为并行与高性能计算、代码分析与优化、大模型推理优化。
E-mail: jwx@bnu.edu.cn



高建花 女, 1995年8月出生于山西省吕梁市。现为北京师范大学人工智能学院讲师。主要研究方向为并行与高性能计算、大模型推理优化。
E-mail: gaojh@bnu.edu.cn