中国科学 信息科学
SCIENTIA SINICA Informationis

Selection of compiler-optimization sequences

高国军, 任志磊, 张静宣, 李晓晨 and 江贺

Citation: 中国科学: 信息科学 **49**, 1267 (2019); doi: 10.1360/N112019-00050

View online: http://engine.scichina.com/doi/10.1360/N112019-00050

View Table of Contents: http://engine.scichina.com/publisher/scp/journal/SSI/49/10

Published by the 《中国科学》杂志社

Articles you may be interested in

畜禽基因组选择研究进展
Chinese Science Bulletin **56**, 2212 (2011);

乙烷选择氧化生成醛类含氧化合物催化剂的研究进展
Chinese Science Bulletin **50**, 729 (2005);

并行推理机RAP/LOP-PIM及其优化并行编译器的设计与实现
Science in China Series A-Mathematics, Physics, Astronomy & Technological Science (in Chinese) **23**, 105 (1993);

A review of selective hydrogenation catalysts on active center and performance
SCIENTIA SINICA Chimica **44**, 1685 (2014);

蛋白质翻译后修饰研究进展
Chinese Science Bulletin **50**, 1061 (2005);

中国科学: 信息科学 2019 年 第 49 卷 第 10 期: 1267–1282

SCIENTIA SINICA Informationis

软件自动化前沿进展专题 • 进展

编译优化序列选择研究进展

高国军[1], 任志磊[1*], 张静宣[2], 李晓晨[1], 江贺[1]

1. 大连理工大学软件学院, 大连 116024
2. 南京航空航天大学计算机与科学技术学院, 南京 210016

* 通信作者. E-mail: zren@dlut.edu.cn

收稿日期: 2019–03–02; 接受日期: 2019–06–06; 网络出版日期: 2019–10–16

国家重点研发计划 (批准号: 2018YFB1003900) 和国家自然科学基金 (批准号: 61722202, 61772107) 资助项目

摘要 在过去的几十年里, 编译器开发者针对各种复杂情况下的编译优化需求, 设计实现了大量的编译优化选项. 在实际开发中, 由编译器提供的标准编译优化序列难以适应复杂场景下待编译程序的编译要求. 一方面, 待编译程序有不同的语义和编译目标, 直接采用标准编译优化序列难以获得理想的优化效果, 若采用不适当的优化序列甚至可能对程序性能等带来负面影响. 另一方面, 随着硬件体系结构的不断发展, 编译环境日益复杂, 编译优化序列亦应进行相应调整. 因此, 如何在错综复杂的优化选项中为待编译程序选择最佳的编译优化序列成为一个具有挑战性的科学问题. 针对上述问题, 研究人员展开了大量的研究, 并取得了诸多成果. 本文旨在归纳编译优化序列选择领域的研究文献, 通过文献搜索, 筛选获得符合条件的 55 篇论文, 从多个视角 (算法、研究类型、目标编译器、基准测试集等) 揭示该领域的研究现状. 通过文献分析可以发现, 当前该领域的主流算法包括两类, 即以遗传算法为代表的启发式搜索算法和以支持向量机为代表的机器学习算法. 超过 80% 的文献的研究类型属于提出解决方案或者实证研究. 在已有的研究中, 实验验证时使用频次最多的编译器和基准测试集分别是 GCC 和 miBench. 本文有助于理解编译优化序列选择领域当前基本进展和发展趋势, 同时为开展该领域研究工作提供了可能的方向.

关键词 编译器, 编译优化序列, 启发式搜索, 机器学习

1 引言

编译器作为软件开发过程中一种重要的基础工具, 是一类将高级计算机语言编写的源程序翻译成目标机器代码的特殊程序. 为了使编译后的目标机器代码具有理想的性能 (如执行速度快、目标代码规模小和功耗低), 编译器提供了大量的编译优化选项. 在软件生产过程中, 开发人员需要决策如何挑选一组编译优化选项形成编译优化序列来编译程序. 通常情况下, 编译器预定义的标准编译优化序列

引用格式: 高国军, 任志磊, 张静宣, 等. 编译优化序列选择研究进展. 中国科学: 信息科学, 2019, 49: 1267–1282, doi: 10.1360/N112019-00050
Gao G J, Ren Z L, Zhang J X, et al. Selection of compiler-optimization sequences (in Chinese). Sci Sin Inform, 2019, 49: 1267–1282, doi: 10.1360/N112019-00050

© 2019《中国科学》杂志社

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

(-O1, -O2, -O3, -Os 等) 会对目标代码某些性能进行优化, 比如目标代码规模、执行速度等, 但是仅仅对编译器开发过程中的测试程序最为有效. 给定一组编译优化序列, 编译器常常会因待编译程序、编译环境和编译目标的不同而产生不同的优化效果, 如若采用不适当的编译优化序列, 甚至可能对程序性能等产生负面影响. 因此, 预定义的标准编译优化序列无法满足复杂多变的实际应用场景. 文献 [1] 表明, 针对不同的待编译程序, 个性化的编译优化序列相对于标准编译优化序列, 可以取得更好的效果. 编译器提供的编译优化选项数量较多, 比如编译器 GCC 有超过 200 个优化选项, LLVM 有超过 100 个优化选项. 对于编译器而言, 从这数量庞大的优化选项中选择合适的选项来组成编译优化序列带来了组合爆炸问题, 通过人工的方式为待编译程序手动选择高效的编译优化序列是一项非常耗时和费力的任务. 因此, 如何利用一些自动化的方法为待编译程序选择高效的编译优化序列 (selection of compiler optimization sequence) 成为重要的科学问题.

由于编译优化序列选择的重要性, 研究者展开了大量研究工作. 随着该领域文献的增多, 阅读和理解文献给该领域相关研究者带来了较大的障碍. 本文通过归纳分析编译优化序列选择相关的文献, 旨在回答以下研究问题: 该领域主要使用了哪些算法来解决编译优化序列选择问题? 从研究类型的角度, 该领域文献是提出了解决方案还是实证研究? 该领域的研究工作主要集中于哪些编译器? 使用了哪些基准测试数据? 该领域中有哪些活跃的研究人员? 围绕这些研究问题, 本文对该领域的研究工作进行归纳, 以帮助更多的研究者快速了解编译优化序列选择的研究现状和发展趋势, 为他们开展相关工作在研究策略、方法技术等方面提供一定的参考.

为了回答上述研究问题, 本文应用系统映射研究方法 [2], 通过不同视角对编译优化序列选择领域相关文献进行归纳, 阐明该领域目前的发展现状. 本文的分析工作主要包括 3 个阶段, 第 1 阶段是确定研究问题以及搜索关键词, 第 2 阶段是收集整理编译优化序列选择领域的文献, 第 3 阶段是综合分析该领域文献研究工作. 第 1 阶段工作中包括以下几个步骤: (1) 确定研究问题; (2) 确定搜索关键词; (3) 确定搜索数据库; 第 2 阶段包括: (1) 使用与研究问题相关的搜索关键词从论文数据库中搜索论文; (2) 筛选搜索得到的论文, 构建用作最终分析的论文库; (3) 抽取论文库中的相关数据用作综合分析. 第 3 阶段工作主要是从 5 个视角 (算法、研究类型、目标编译器、基准测试集和作者统计) 分析文献, 并回答第 1 阶段中提到的一系列研究问题.

本文的主要贡献如下: (1) 构造搜索关键词, 从数据库中搜索筛选了 55 篇关于编译优化序列选择的文献, 并构建一个论文库用于综合分析, 帮助研究者快速了解该领域; (2) 通过分析编译优化序列选择领域文献, 整理了该领域使用的算法, 以及该领域文献的研究类型分布情况; (3) 分析了在编译优化序列选择领域研究人员倾向使用的编译器和基准测试数据, 以及该领域发表论文频次较高的作者.

本文的组织结构如下: 第 2 节介绍与本文相关的编译优化序列选择背景知识; 第 3 和 4 节分别介绍本文中论文库的搜索筛选构建过程和该领域论文综合分析结果; 第 5 节给出论文结论和进一步工作展望.

2 背景知识

本节讨论编译优化序列选择的相关背景知识.

如图 1 所示, 编译器以源程序作为输入, 经过词法分析、语法分析、语义分析、中间代码生成、代码优化、目标代码生成, 最终得到可执行的目标机器代码 [3]. 根据功能不同, 编译器的组成部分可划分为前端 (front end) 与后端 (back end). 其中, 前端包括词法分析、语法分析、语义分析和中间代码生成, 其功能是解析源程序, 并将之转换为某种中间表示 (intermediate representation). 后端则包括代码

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

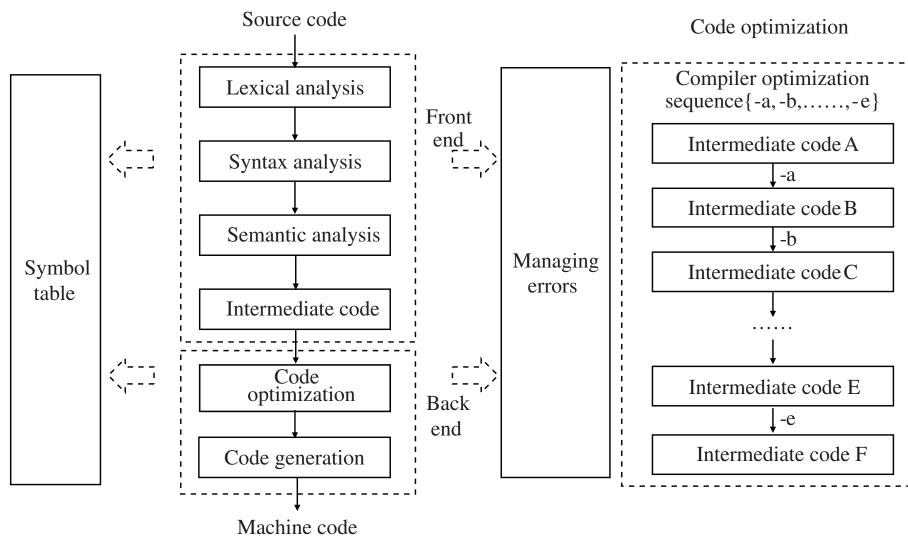

图 1 编译器结构

Figure 1 The structure of compiler

优化和目标代码生成, 负责生成面向特定机器的目标机器代码, 其中图 1 中右半部分展示了代码优化过程中编译器采用优化序列 {-a, -b, . . ., -e} 对待编译程序的优化过程. 此外, 编译器的符号表管理部分负责管理编译过程中源程序的各种信息, 而出错管理部分负责记录源程序出错时错误的类别和出错地点. 比较常见的编译器包括 GCC (GNU compiler collection), LLVM 和 Open64 等.

现代编译器提供了大量的编译优化选项, 以达到提高目标机器代码的执行速度、降低代码规模等编译目标. 但是, 使用穷举搜索在数量庞大的选项组合中选择最优的编译优化序列是不现实的. 假设编译优化选项个数为 20, 其穷举搜索时搜索空间大小则为 $2^{20} = 1048576$. 考虑到编译器 GCC 提供了超过 200 个编译优化选项, 其搜索空间的大小将呈指数级增长. 因此, 编译器提供了一些预定义的标准编译优化序列. 这些优化序列 (如 -O1, -O2, -O3 等) 是将编译器提供的编译优化选项进行组合以达到一定的编译目标, 比如提升目标代码的执行速度、减小目标代码的规模等. 以编译器 GCC4.9.4 为例, 其提供了一系列标准编译优化序列.

(1) -O0: 该选项不做任何优化, 是 GCC 编译器的默认优化选项.

(2) -O1: 该优化序列采用一些优化算法降低代码规模和可执行代码的运行时间.

(3) -O2: 该优化序列会牺牲部分编译速度以进行更多的优化, 是比 -O1 更高级的编译优化序列. -O2 将执行几乎所有的不包含时间和空间折中的优化. 与 -O1 比较而言, -O2 以更多编译时间为代价, 提高了生成代码的执行效率.

(4) -O3: 该优化序列比 O2 更进一步地进行优化, 降低目标代码的执行时间.

(5) -Os: 该优化序列主要是对代码的规模进行优化. 打开了大部分 -O2 优化中不会增加代码规模的优化选项, 并对代码的规模做更深层的优化.

(6) -Ofast: 该优化序列不会严格遵循语言标准, 除了启用所有的 -O3 优化选项之外, 也会针对某些语言启用部分优化, 如: -ffast-math.

除了编译器提供的标准编译优化序列, 其余各个优化选项的实际功能需要开发人员详细阅读编译

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

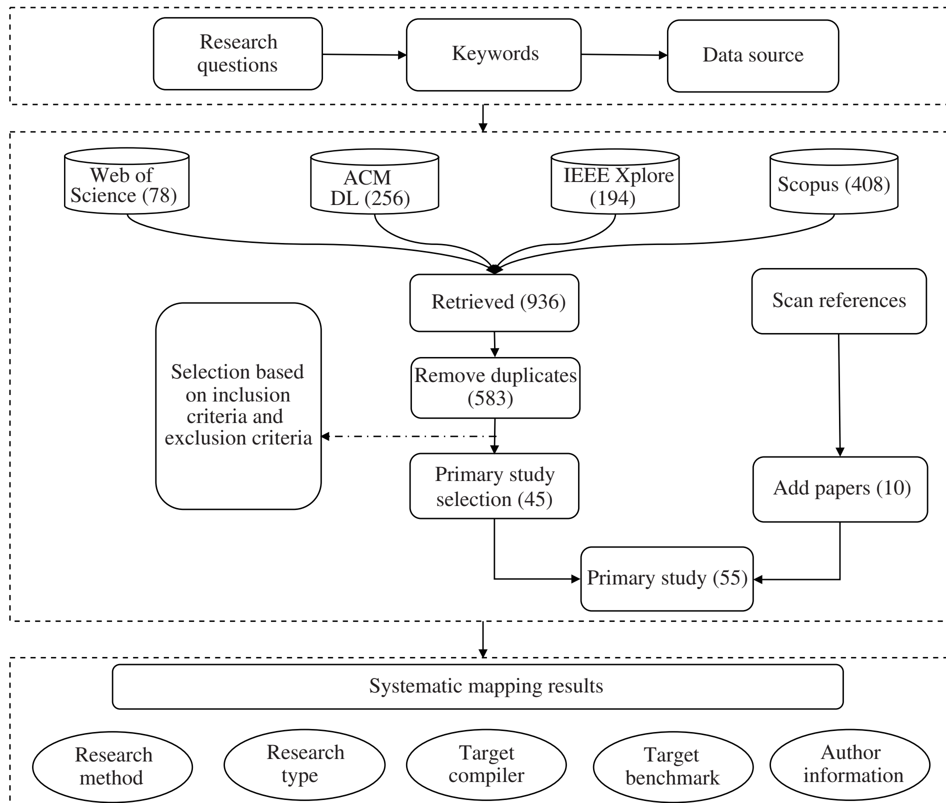

图 2　文献分析流程

Figure 2　Procedure of publication analysis

器相关文档, 以此来决定实际开发过程中选择哪些优化选项, 如 GCC [1)]和 LLVM [2)]. 然而, 标准编译优化序列难以满足复杂多变的实际应用场景. 面对特定的待编译程序, 很多研究者探索如何在有限的时间内获得高效的编译优化序列.

3　文献分析方法

本节详细阐述编译优化序列选择领域文献的分析方法. 本文参考文献 [2] 以及一些软件工程领域的系统映射研究工作 [4~7] 设计文献分析流程, 如图 2 所示, 分析流程包括 3 个阶段: 分别是准备阶段、论文库构建阶段、综合分析阶段.

在准备阶段, 首先, 根据本文需要映射研究的编译优化序列选择领域确定研究问题 (research questions). 然后, 通过分析这些研究问题确定搜索关键词 (keywords) 和需要搜索的论文数据库 (data source). 在论文库构建阶段, 利用确定好的搜索关键词在数据库中搜索该领域的研究论文组成初始论文库, 再经过去重, 使用一定的选择标准筛选符合要求的论文. 之后, 通过这些论文的参考文献添加没有出现在搜索结果中的论文, 最终组成用来分析该领域研究工作的论文库. 综合分析阶段是在构建好论文库之后, 通过综合分析相关文献, 从 5 个视角 (算法、研究类型、目标编译器、基准测试集、作

1) GCC's options that control optimization. https://gcc.gnu.org/onlinedocs/gcc-9.1.0/gcc/Optimize-Options.html#Optimize-Options.

2) LLVM's analysis and transform passes. http://llvm.org/docs/Passes.html.

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

者信息) 来展现该领域的概况.

3.1 研究问题

在系统映射研究中, 研究问题能够对整个研究过程起到引导的作用. 首先, 在确定搜索策略中的搜索串中提到的关键词主要来自于研究问题. 其次, 在论文选择以及排除过程中也需要基于研究问题. 第三, 从论文中提取数据必须与研究问题有关. 第四, 最后分析提取数据也要基于能够解答研究问题的目的. 本文的研究问题如下:

RQ1. 编译优化序列选择问题研究有哪些算法? 这些算法可以如何分类? 这些算法是针对什么编译目标 (执行速度、代码规模等) 提出的?

RQ2. 按照论文研究类型, 现有研究工作如何分类? 该领域论文研究类型呈怎样的分布? 文献 [2] 在提出的系统映射研究指导方案中, 将论文划分为 6 类: 实证研究 (validation research)、评价研究 (evaluation research)、提出解决方案 (solution proposal)、哲理性研究 (philosophical papers)、个人观点 (opinion papers)、经验研究 (experience papers). 本文也将论文库中的论文分为这 6 类, 从论文研究类型的视角分析编译优化序列选择领域.

RQ3. 编译优化序列选择问题针对哪些编译器展开研究? 这个研究问题是从编译器主体这个视角分析编译优化序列选择领域.

RQ4. 编译优化序列选择问题中使用了哪些基准测试集 (benchmark)? 这个研究问题是从基准测试数据的视角来分析编译优化序列选择领域.

RQ5. 编译优化序列选择研究工作中哪些作者比较活跃?

3.2 文献搜索策略

文献搜索部分是通过分析映射研究涉及的范围和确定的研究问题, 抽取一些关键词组成搜索关键词, 然后利用搜索关键词从数据库中搜索相关论文组成初始论文库.

本文研究的领域为编译优化序列选择问题. 通过分析, 我们确定搜索关键词为 "compiler + optimization + (sequence or exploration)". 经过浏览已知的一些该领域论文可以发现, 这几个关键词经常出现并且覆盖面比较广, 有助于从数据库中搜索到该领域论文. 根据文献 [2] 中提到的映射研究数据源确定的方式, 以及在软件工程领域其他系统映射研究 [4~7] 中经常使用的数据库, 本文确定使用比较相关的几个数据库作为数据来源, 包括 Web of Science, ACM Digital Library, IEEE Xplore 和 Scopus.

确定好搜索关键词和数据源之后, 本文使用搜索关键词在确定好的数据库中搜索论文题目、摘要和关键词. 最终, 从 Web of Science, ACM Digital Library, IEEE Xplore 和 Scopus 4 个数据库分别搜索获得论文 78, 256, 194 和 408 篇, 总计论文数量为 936 篇, 去重后得到的初始论文库包含 583 篇论文.

3.3 文献筛选过程

文献筛选是将筛选规则应用于初始论文库上, 剔除不符合标准的论文, 并补充遗漏的相关论文, 以此得到编译优化序列选择领域最相关的论文组成最终的论文库, 用作后续的综合分析.

本文的筛选标准主要由两部分组成 [2]: 包含标准 (inclusion criteria) 和排除标准 (exclusion criteria). 其中包含标准包括: (1) 论文是否属于软件工程领域? (2) 论文是否与编译优化有关? (3) 论文是否与编译优化序列选择有关? 排除标准包括: (1) 论文页数是否少于 4 页? (2) 论文是否属于英文论文? (3) 论文是否提供全文下载?

通过浏览阅读论文的题目、摘要、关键词以及正文, 将筛选标准应用于初始论文库中的 583 篇论

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

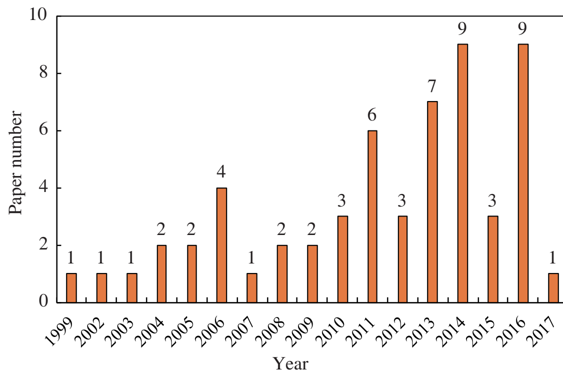

图 3 (网络版彩图) 每年发表论文数

Figure 3 (Color online) The number of publications per year

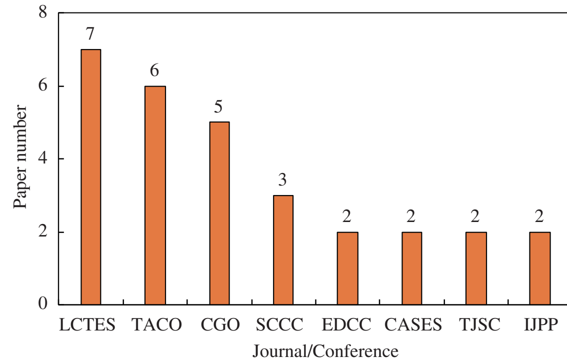

图 4 (网络版彩图) 论文发表期刊/会议

Figure 4 (Color online) Venues of publications

表 1 论文发表期刊/会议全称

Table 1 Full names of publication venues

Abbreviation	Full name of journals/conferences
LCTES	ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems
TACO	*ACM Transactions on Architecture and Code Optimization*
CGO	IEEE/ACM International Symposium on Code Generation and Optimization
SCCC	International Conference of the Chilean Computer Science Society
EDCC	European Dependable Computing Conference
CASES	International Conference on Compilers, Architecture, and Synthesis for Embedded Systems
TJSC	*The Journal of Supercomputing*
IJPP	*International Journal of Parallel Programming*

文, 最终得到 45 篇论文. 比如文献 [8] 正文只有 3 页, 仅仅简单描述了解决方案的算法, 并没有详细介绍, 因此没有被收录到论文库中. 文献 [9] 提出的解决方案是一种优化目标生成代码的算法, 而不是选择编译优化序列, 与本文所分析的研究领域不符, 因此也没有收录到论文库中. 之后我们通过余下论文的参考文献又添加了 10 篇不在搜索结果中的论文, 最终组成包含 55 篇论文的论文库, 用作后续对编译优化序列选择领域的综合分析.

图 3 是论文库中论文发表年份的分布情况. 横坐标表示论文发表年份, 纵坐标表示发表论文数量. 我们可以发现, 在 2005 年之前编译优化序列选择领域论文数量相对较少, 之后发表论文的数量有了一定的提升, 有更多的研究者针对编译优化序列选择问题提出了解决方案, 这表明这些年来软件工程领域研究者对于编译优化序列选择问题的关注和研究逐步增加, 越来越多的研究人员致力于提出更好的算法来解决该问题, 反映了该领域近期研究成果显著. 本文针对该领域相关论文的整理分析, 将会为潜在研究者提供一定的帮助指导.

图 4 是论文库中论文发表期刊/会议的分布情况. 横坐标表示论文发表的期刊和会议, 纵坐标表示对应期刊会议的论文数量, 该图中列出了发表至少两篇论文的期刊和会议. 表 1 是图 4 中所列期刊/会议的全称. 可以发现, 期刊 TACO 和会议 LCTES, CGO 上发表编译优化序列选择领域的论文最多, 分别是 7, 6, 5 篇.

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

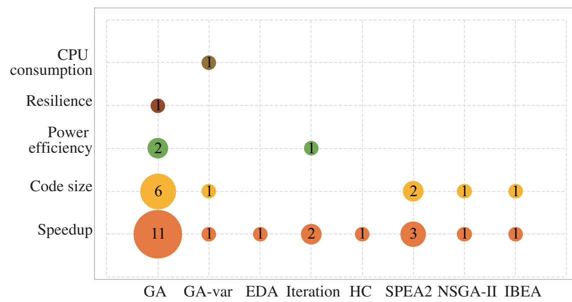

图 5 (网络版彩图) 启发式算法及优化目标统计结果

Figure 5 (Color online) Heuristic search algorithms and objectives

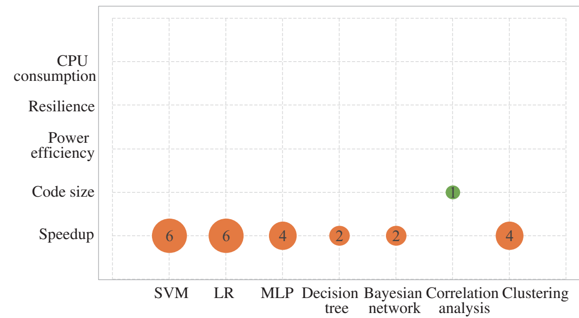

图 6 (网络版彩图) 机器学习算法及优化目标统计结果

Figure 6 (Color online) Machine learning algorithms and objectives

3.4 数据提取

为了回答上述提出的几个研究问题, 我们需要从构建好的论文库中提取相关信息. 这些信息包括: 论文提出的算法、针对的编译目标、研究类型、使用的编译器、论文实验验证时使用的基准测试集、论文的作者信息. 其中研究类型根据文献 [2] 分为以下 6 类:

(1) 实证研究: 研究工作提出的算法是比较新的并且还没有在实际中实现, 仅仅证明在实验室环境下有效.

(2) 评价研究: 研究工作对于实际中已经实现的算法进行评价总结, 展示实现细节以及其优缺点.

(3) 提出解决方案: 针对某一个问题提出了一种较新的解决方案或者在已有技术上的重大扩展.

(4) 哲理性研究: 通过以一种新的方法或者概念框架来看待某一个问题.

(5) 个人观点: 阐述作者对于某些问题或者技术的个人看法, 如某项技术是更好或者更差的.

(6) 经验研究: 作者以个人经验来解释某些问题或者算法.

4 分析结果

本节主要从多个视角综合分析编译优化序列选择领域, 给出该领域的研究概况.

4.1 算法视角

从算法视角分析编译优化序列选择领域相关研究时, 主要包含两个方面: 其一是该领域的论文涉及的算法, 其二是相关研究工作的优化目标.

4.1.1 算法

首先, 该领域中使用的算法主要包括启发式搜索算法和机器学习算法两类. 图 5 和 6 分别统计了论文库中涉及的各种启发式搜索算法和机器学习算法的使用情况. 其中横坐标表示该领域使用的各类算法, 纵坐标表示论文库中文献涉及到的编译优化目标. 图中圆的大小表示算法优化某个编译目标的论文数量, 圆中的数字给出了具体数值. 例如图 5 中左下角标注 11 的圆, 它表示使用遗传算法且优化目标是目标代码加速比的论文数量为 11 篇. 从图中可以看出, 在启发式搜索算法中使用最广泛的优化算法是遗传算法, 有 20 篇相关文献, 在整个论文库中的占比超过了三分之一. 一些研究工作也使用遗传算法的一些变种以及爬山算法、分布估计算法等来选择编译优化序列.

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

启发式搜索算法使用启发式的方法在编译优化选项组合空间中搜索最优的编译优化序列. 比如, Cooper 等 [10] 使用随机搜索策略 (randomized search algorithm) 从 16 个优化选项中选择长度为 10 的编译优化序列来编译测试程序, 并验证了这种方法可以发现比预定义的标准编译优化序列效果更好的选项组合. Kulkarni 等 [11] 提出了一个名为 "VISTA" 的交互编译系统, 使用遗传算法 (genetic algorithm) 和人工辅助引导结合的方法搜索最优的编译优化序列.

遗传算法是一种模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型, 其通过模拟自然进化过程搜索最优解. 这种算法从针对问题解的群体开始, 并对其进行进化和优化, 其中群体中的每个个体是问题解的编码表示, 遗传算法通过使用包括选择、交叉和变异的优化操作对种群进行迭代进化得到更好的解. 在编译优化序列选择领域中, 遗传算法得到了广泛的使用.

Jantz 等 [12] 使用遗传算法为 JIT 编译器选择最优的编译优化序列, 并且取得了不俗的结果. Ansel 等 [13] 提出了一个为程序满足多个目标而选择编译优化序列的开源框架 "OpenTuner", 该框架使用了包括遗传算法在内的多种进化算法, 其在 6 个项目和 16 个基准测试集上的实验表明该框架效果良好, 能够获得最高 2.8 倍的加速比. Boussaa 等 [14] 在使用遗传算法的基础上, 利用一种新的指标来表示种群中个体的稀疏性, 在每次迭代选择时尽量确保种群中个体的多样性, 进而为待编译程序选择合适的编译优化序列.

除此之外, 为了在目标代码执行速度和代码规模等优化目标之间有所权衡, 研究人员使用一些多目标优化算法来选择待编译程序对应的编译优化序列. Lokuciejewski 等 [15, 16] 分析了多目标优化算法在编译优化序列选择中的应用情况, 其使用 SPEA2, NSGA-II 和 IBEA 为待编译程序选择满足目标代码执行速度、规模等目标的编译优化序列. 实验表明这些多目标优化算法相比于标准优化序列获得了显著的提升. Chebolu 和 Wankar [17] 使用基于带权重的价值函数的遗传算法来为待编译程序搜索高效的编译优化序列. 在基准测试集 SPEC 2006 上的实验结果表明, 该算法可以获得比 "-Ofast" 生成的目标代码执行时间更快或者相同的编译优化序列, 同时目标代码的规模不超过 "-Os". 该算法搜索到的编译优化序列生成的目标代码可以在执行速度和代码规模两个目标上达到一个平衡. 文献 [13] 介绍了一个为待编译程序构造多目标优化的编译优化序列的开源框架 "OpenTuner". 该框架支持用户自定义设置多种搜索方法 (包括遗传算法、粒子群算法、随机搜索算法等), 针对多个目标 (比如执行时间、功耗、代码规模等) 为待编译程序搜索合适的编译优化序列.

虽然启发式搜索算法能够产生高效的编译优化序列, 但是需要大量的时间来运行整个迭代过程. 因此, 研究人员开始尝试使用机器学习算法来进行编译优化序列的选择. 首先, 在线下使用一系列静态特征 (如代码行数、代码中循环次数等) 或者动态特征 (如代码运行过程中读写操作的数量、浮点数运算统计等) 表征源程序. 其次, 构造训练数据集并训练预测模型. 训练数据中每个实例都是一个三元组 $\langle F, O, R\rangle$. 其中 F 是待编译程序的特征表示, O 是待编译程序编译时使用的编译优化序列, R 是待编译程序使用该编译优化序列编译后的效果. 在训练完成后, 该模型就可以用于预测不同编译优化序列对于待编译程序的编译效果. 比如, Cavazos 等 [18] 使用代码运行时特征来表征待编译程序, 以此来训练逻辑回归模型. 该方法在基准测试集 SPEC 2000 和 miBench 上的效果良好, 在目标代码的执行速度上相对于预定义的标准编译优化序列 -Ofast 取得 17% 的提升. 文献 [19] 介绍了世界上第 1 个基于机器学习的开源编译器 "Milepost GCC", 这是一个经过对 GCC4.4 进行修改形成的模块化、可扩展的编译器, 支持对待编译程序进行静态特征抽取, 训练机器学习模型, 并预测编译优化序列的编译效果. 该工具为基于机器学习的编译优化序列选择研究提供了极大方便, 之后有许多相关研究 [20~22] 都是在该工具的基础上完成的. 此外, Ashouri 等 [23] 分析了编译器 LLVM 中优化选项之间的依赖关系, 利用程序动态特征来训练 Bayes 网络. 然后使用该模型预测下一阶段应该出现的优化选项, 直到预测完成

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

表 2 启发式算法与机器学习的对比

Table 2 The comparison between heuristic search and machine learning

	Heuristic algorithms	Machine learning
Input	Encode	Feature representation
Output	Compiler optimization sequence	Compiler optimization sequence
Training set	Unnecessary	Necessary
Time consuming	Long	Short

为止, 最终获得待编译程序对应的高效编译优化序列.

如图 6 所示, 我们发现在机器学习算法中使用比较广泛的是支持向量机 (support vector machine, SVM) 和线性回归 (linear regression, LR), 两种算法均在 6 篇文献的研究工作中被使用. 此外, 编译优化序列选择领域中还使用了多层感知机 (multilayer perceptron, MLP)、决策树 (decision tree)、Bayes 网络 (Bayesian network)、关联分析 (correlation analysis) 和聚类算法.

支持向量机是一种建立在统计学习理论的 VC 维理论和结构风险最小原理基础上的机器学习算法, 通过构造超平面来实现分类和回归分析. Park 等 [24] 应用代码执行中的动态特征, 使用支持向量机来训练预测两个编译优化序列的优劣, 进而为待编译程序选择更优的编译优化序列. Park 等也使用基于图的代码特征 [22] 以及自定义的模式特征 [24], 并通过支持向量机来训练学习模型, 预测编译优化序列对于待编译程序的加速比, 从而获得理想的结果.

线性回归是该领域使用频次较高的另一种学习算法. 线性回归是利用数理统计中的回归分析确定变量之间关系的分析方法, 是一种广泛使用的监督学习算法. Ashouri 等 [23] 利用代码动态特征, 并分析了编译优化选项之间的依赖关系, 使用线性回归算法为待编译程序预测高效的编译优化序列, 获得的平均加速比为 1.31. 另外, 聚类算法这种非监督学习算法也被应用于编译优化序列选择问题. Martins 等 [25] 利用一定的转换规则将待编译程序进行 DNA 编码序列, 利用近邻结合法 (neighbor joining algorithm) 构造图结构, 并通过聚类算法选择与待编译程序相似的程序对应的编译优化选项, 从而获得高效的编译优化序列.

表 2 列出编译优化序列选择领域中启发式算法和机器学习算法的对比情况. 应用启发式算法时需要将待编译程序转化为对应的编码表示, 机器学习算法则需将待编译程序的特征表示作为输入. 同时由于启发式算法会在整个搜索空间内搜索最优解, 而机器学习算法只能通过模型获取一定量编译优化序列的效果, 并不能保证发现最优的编译优化序列. 此外, 由于搜索空间的庞大造成启发式算法需要花费较机器学习算法更多的时间搜索到最优编译优化序列. 尽管机器学习算法用时更少, 但是需要一定量数据来训练学习模型, 而启发式算法并不需要. 因此, 启发式算法适用于没有训练数据集且对算法运行时间要求不严格的场景, 而机器学习算法适用于有训练数据集或者对算法运行时间要求严格的情况.

4.1.2 编译优化目标

从算法视角分析编译优化序列选择领域时, 本文关注的第 2 方面是编译优化目标. 如图 5 和 6 所示, 大多数文献是针对目标机器代码的执行速度 [23,25,26] 开展研究工作的. 在图 5 启发式搜索算法中有 21 篇文献是针对目标代码加速比, 占到这部分文献的一半以上. 图 6 显示, 机器学习算法中将目标代码加速比作为优化目标的文献占到绝大多数, 占到这部分文献的 95% 以上. 另一个受到研究人员关注的优化目标是目标代码的规模 [27,28]. 目标代码规模对于程序编译是一个非常重要的优化目标, 尤

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

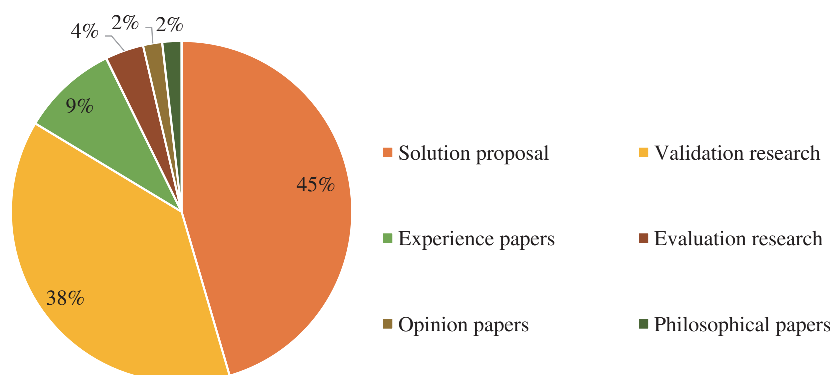

图 7　(网络版彩图) 研究类型统计结果

Figure 7　(Color online) Statistical results of research types

其是在当前嵌入式程序广泛应用的情况下, 尽可能减少存储空间可以带来显著的效益. 论文库中有 12 篇文献使用了遗传算法、关联分析等为待编译程序选择满足该目标的编译优化序列.

除了将目标代码的执行速度和代码规模作为优化目标之外, 研究者还分析了 CPU 使用 (CPU consumption) [14]、功耗 (power efficiency) [29] 以及差错恢复能力 (resilience) [30] 等目标, 以满足特定情况下某些程序的需求. 比如, Sangchoolie 等 [31] 分析了在使用编译器 GCC 提供的各个标准编译优化序列 (-O0, -O1, -O2, -O3 等) 编译待编译程序的差错恢复能力. 实验结果表明在使用标准编译优化序列时, 待编译程序的差错恢复能力没有明显的区别. Narayanamurthy 等 [30] 使用遗传算法为需要对硬件错误具有较强恢复能力的程序选择编译优化序列. 实验表明, 与标准编译优化序列相比, 遗传算法发现的编译优化序列所获得的目标代码具有更高的差错恢复能力.

4.2　研究类型视角

根据文献 [2], 本文按照研究类型将论文库中的文献分为 6 类: 实证研究、评价研究、提出解决方案、哲理性研究、个人观点、经验研究. 图 7 是本文分析的编译优化序列选择领域中论文研究类型的分布情况. 根据论文数量排序结果为: 提出解决方案 (25 篇)、实证研究 (21 篇)、经验研究 (5 篇)、评价研究 (2 篇)、个人观点 (1 篇)、哲理性研究 (1 篇). 如图 7 所示, 该领域绝大多数论文属于提出解决方案和实证研究, 两者占比超过 80%, 其余类型的研究比例相对较少. 这表明该领域的研究工作主要集中于提出算法来解决编译优化序列选择问题或者验证某些方案的有效性.

提出解决方案是针对该问题提出一种较新的解决方案, 比如, 文献 [25] 提出了一种新方法, 其使用 DNA 编码表示待编译程序, 并利用聚类算法来为待编译程序寻找相似程序对应的优化选项, 进而选择最优的编译优化序列. 实证研究是验证所提方法的有效性, 如 Ashouri 等 [26] 验证了基于 Bayes 网络框架的有效性, 该框架在基准测试集 cBench 上能够获得 1.48 的加速比. 经验研究是对某些问题的分析, 例如文献 [32] 分析了静态特征、动态特征等对于机器学习算法在应用于编译优化序列选择时的影响. 评价研究是对已实现的方法、框架评价总结, 文献 [14, 19] 分别总结分析了两个编译优化序列选择框架 NOTICE [19] 和 Milepost GCC [14]. 个人观点是指作者对于某些问题的个人想法, 如 Nazarian 等 [33] 通过总结分析优化能量消耗和优化程序可靠性两类研究工作中的各类方法, 针对编译优化能量消耗和程序可靠性的兼容性提出了自己的观点. 分析结果指出有两种方法组合在上述优化两个目标时兼容性更佳, 包括指令重调度和指令重复、循环展开和可执行断言. 哲理性研究表示作者通过一种新的方式来看待该问题, 如 Touati 等 [34] 通过形式化的方法证明了编译优化选项排序以及编译优化参数

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

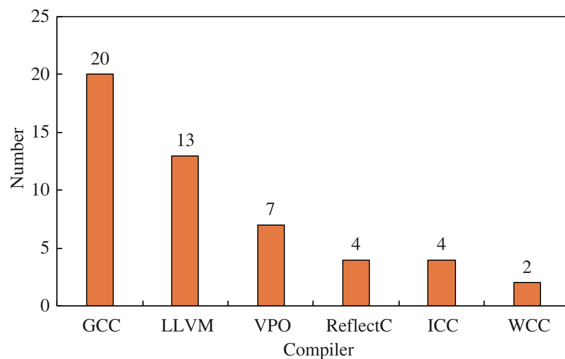

图 8 (网络版彩图) 目标编译器统计结果

Figure 8 (Color online) Statistical results of target compilers

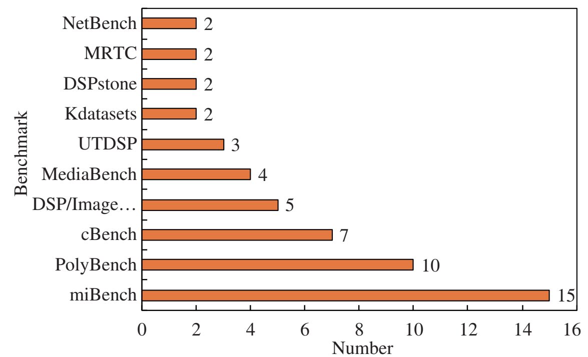

图 9 (网络版彩图) 基准测试集统计结果

Figure 9 (Color online) Statistical results of target benchmarks

调优是不可判定的, 即在迭代编译时无法找到针对这两个问题的最优解.

4.3 目标编译器视角

编译优化序列选择涉及到众多的编译器, 包括 GCC [26], LLVM [23], VPO [35] 等. 图 8 给出了论文库中研究者所使用的编译器统计结果, 按照编译器 GCC, LLVM, VPO, ReflectC, ICC, WCC 使用次数排序结果为 20, 13, 7, 4, 4, 2. 其中使用频次较高的编译器为 GCC 和 LLVM.

编译器 GCC 是由 GNU 开发的编程语言编译器, 以 GPL 许可证发行. GCC 是 GNU 项目的关键部分, 也是 GNU 工具链的主要组成部分之一. 它原本只能处理 C 语言代码. 在发布后很快得到扩展, 进一步可以处理 C++, Fortran, Pascal, Objective-C, Java, Ada, Go 及其他语言. 文献 [26,32,36] 均是基于 GCC 编译器来为待编译程序选择编译优化序列, 基于机器学习的开源编译器 "Milepost GCC" [19] 也是在 GCC 的基础上发展而来.

其次, 编译器 LLVM 是一个 C++ 编写的编译器和工具技术的集合, 其优点包括高度模块化的现代化设计、语言无关的中间表示以及众多用于优化的工具和函数. 文献 [23,25,30,37] 均是基于编译器 LLVM 的研究工作, 它们利用进化算法或者机器学习算法为待编译程序选择最优的编译优化序列并验证算法的有效性.

4.4 目标基准测试集视角

为了验证研究工作中提出的算法的有效性, 需要使用一些待编译程序组成基准测试集, 在此基础上验证所提出方案的效果. 图 9 是编译优化序列选择领域中基准测试集使用频次的统计结果, 其使用次数均超过 1 次. 我们可以看出, 使用频次排序前 5 的基准测试集以及其使用频次分别是 miBench (15), PolyBench (10), cBench (7), DSP/Image benchmark (5), MediaBench (4).

其中基准测试集 miBench 是一个开源的嵌入式基准测试程序组, 包含汽车电子、消费类电子、网络、办公自动化、信息安全和电信等 6 类程序, 所有程序均采用 C 语言编写, 但是仅仅提供了基于编译器 GCC 的可供参考的编译优化序列. 而基准测试集 cBench 是在 miBench 的基础上, 由研究人员补充了编译器 LLVM, Open64 等可供参考的编译优化序列. 在实验验证时, 这些编译优化序列和相关的待编译程序被用作基准来评价各种算法的优劣 [23,38].

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

表 3　作者及其论文发表频次

Table 3　Authors and their publication frequencies

ID	Author	Frequency
1	John Cavazos	9
2	Prasad A. Kulkarni	7
3	David Whalley	6
4	Amir Hossein Ashouri	5
5	Cristina Silvano	5
6	Eunjung Park	5
7	Gianluca Palermo	5
8	Jack Davidson	5
9	Joao M. P. Cardoso	5
10	Keith D. Cooper	5
11	Ricardo Nobre	5

4.5　作者视角

本文从编译优化序列选择领域的论文库中收集了论文的作者信息, 通过统计分析来获取该领域发表论文数量较多、影响力较大的作者. 论文库中包含 55 篇论文, 共涉及 137 位不同的作者, 平均每篇论文作者数量为 4.24 位, 大多数论文的作者数量为 1~6 位. 表 3 给出了论文发表频次超过 4 次的作者信息及其论文发表频次统计结果. 从表 3 中可以看出, 论文发表频次超过 4 次的共有 11 位作者. 其中, John Cavazos 是排名第 1 位的作者, 其来自于美国特拉华大学 (University of Delaware), 在该领域发表 9 篇论文. 该作者主要致力于使用机器学习算法来为待编译程序选择最优的编译优化序列 [23, 26]. 排在第 2 位和第 3 位的分别是 Prasad A. Kulkarni 和 David Whalley, 在该领域分别发表了 7 篇和 6 篇论文. 表 3 中其余 8 位作者均发表了 5 篇论文, 其中 Amir Hossein Ashouri, Cristina Silvano, Eunjung Park 主要是使用不同类型的代码特征 (静态特征、动态特征等) 来应用机器学习算法解决问题, 而 Keith D. Cooper 主要是使用迭代编译的思想来为该领域问题提供解决方案.

值得注意的是, 中国学者在该领域也做出了出色的工作. 比如中国科学院计算技术研究所吴承勇教授团队分析了编译优化序列选择中迭代编译方法的有效性 [39]. 他们通过 GCC 和 ICC 两个编译器在 32 个程序上的编译实验, 发现最优的编译优化序列与待编译程序有一定相关性, 并且加速比最高达到 3.75. 国防科技大学王正华教授团队也提出了一种在高性能计算领域有效的迭代编译搜索算法 [40], 该算法能够在较短的迭代次数内为待编译程序找到高效的编译优化序列.

5　结束语

待编译程序、编译环境和编译目标的多样复杂性, 使得编译器提供的标准编译优化序列无法满足所有的编译需求. 面对编译器大量的优化选项, 快速有效的为待编译程序确定高效的编译优化序列成为重要的科学问题, 受到越来越多研究者的关注.

本文通过综合分析编译优化序列选择领域文献, 从多个视角来深入分析该领域的发展现状.

(1) 该领域使用的算法主要是启发式搜索算法 (GA, SPEA2, NSGA-II 等) 以及机器学习算法 (SVM, LR, MLP 等);

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

(2) 该领域研究工作的研究类型主要是提出解决方案和实证研究, 两者占比超过 80%;

(3) 该领域针对的目标编译器主要是 GCC 和 LLVM;

(4) 该领域实验验证时使用的基准测试集主要是 miBench, PolyBench 和 cBench;

(5) 该领域发表论文频次较高的作者分别是 John Cavazos, Prasad A. Kulkarni 和 David Whalley.

在实际开发应用中, 编译目标日益多样化. 比如, 除了目标代码的执行速度、代码规模等编译目标外, 目标代码的安全性、可靠性等也变得日益重要. 面对多样化的编译优化目标, 如何设计更加高效的算法来求解编译优化序列选择问题是下一阶段的研究重点. 从启发式搜索算法的方面看, 目前研究者仅尝试了一些经典的启发式搜索算法 (如遗传算法等). 在未来的工作中, 研究者可以尝试一些新的高效搜索算法, 比如使用多级搜索空间约简方法 [41, 42] 引导搜索过程, 减少算法的迭代过程等. 从机器学习算法的方面看, 使用编译器优化选项的相关文档说明增强待编译程序的特征表示 [43, 44], 提高机器学习模型准确度, 此外, 各种深度学习技术已经推动 Bug 报告摘要 [45]、代码生成 [46] 等领域取得许多新的突破, 也为编译优化序列选择研究提供了新的方向和可能.

参考文献

1 Hoste K, Eeckhout L. Cole: compiler optimization level exploration. In: Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization, 2008. 165–174

2 Petersen K, Feldt R, Mujtaba S, et al. Systematic mapping studies in software engineering. In: Proceedings of International Conference on Evaluation & Assessment in Software Engineering, 2008. 68–77

3 Chen Y Y, Zhang Y. Compiler Principle. 3rd ed. Beijing: Higher Education Press, 2014

4 Garousi V, Mesbah A, Betin-Can A, et al. A systematic mapping study of web application testing. Inf Softw Tech, 2013, 55: 1374–1396

5 Neto C R L, Neto P A M S, Almeida E S D, et al. A mapping study on software product lines testing tools. In: Proceedings of the 24th International Conference on Software Engineering and Knowledge Engineering, 2012. 628–634

6 Ouhbi S, Idri A, Fernández-Alemán J L, et al. Software quality requirements: a systematic mapping study. In: Proceedings of the 20th Asia-Pacific Software Engineering Conference, 2013. 231–238

7 Cosentino V, Canovas Izquierdo J L, Cabot J. A systematic mapping study of software development with GitHub. IEEE Access, 2017, 5: 7173–7192

8 de Lima E D, Silva A F D, Herrera C. A case-based reasoning approach to find good compiler optimization sequences. In: Proceedings of the 32nd International Conference of the Chilean Computer Science Society (SCCC), Temuco, 2013. 8–10

9 Yi Q, Wang Q, Cui H M. Specializing compiler optimizations through programmable composition for dense matrix computations. In: Proceedings of IEEE/ACM International Symposium on Microarchitecture, 2014. 596–608

10 Cooper K D, Grosul A, Harvey T J, et al. Exploring the structure of the space of compilation sequences using randomized search algorithms. J Supercomput, 2006, 36: 135–151

11 Kulkarni P, Zhao W, Moon H, et al. Finding effective optimization phase sequences. In: Proceedings of ACM Sigplan Conference on Language, Compiler, and Tool for Embedded Systems, 2003. 12–23

12 Jantz M R, Kulkarni P A. Performance potential of optimization phase selection during dynamic JIT compilation. ACM SIGPLAN Not, 2013, 48: 131–142

13 Ansel J, Kamil S, Veeramachaneni K, et al. Opentuner: an extensible framework for program autotuning. In: Proceedings of the 23rd International Conference on Parallel Architectures and Compilation, 2014. 303–316

14 Boussaa M, Barais O, Baudry B, et al. NOTICE: a framework for non-functional testing of compilers. In: Proceedings of IEEE International Conference on Software Quality, Reliability and Security, 2016. 335–346

15 Lokuciejewski P, Plazar S, Falk H, et al. Multi-objective exploration of compiler optimizations for real-time systems. In: Proceedings of IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, 2010. 115–122

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

16 Lokuciejewski P, Plazar S, Falk H, et al. Approximating Pareto optimal compiler optimization sequences-a trade-off between WCET, ACET and code size. Softw-Pract Exper, 2011, 41: 1437–1458

17 Chebolu N A B S, Wankar R. Multi-objective exploration for compiler optimizations and parameters. In: Proceedings of International Workshop on Multi-disciplinary Trends in Artificial Intelligence, 2014. 23–34

18 Cavazos J, Fursin G, Agakov F, et al. Rapidly selecting good compiler optimizations using performance counters. In: Proceedings of the International Symposium on Code Generation and Optimization, 2007. 185–197

19 Fursin G, Kashnikov Y, Memon A W, et al. Milepost GCC: machine learning enabled self-tuning compiler. Int J Parallel Prog, 2011, 39: 296–327

20 Nobre R, Martins L G A. Use of previously acquired positioning of optimizations for phase ordering exploration. In: Proceedings of International Workshop on Software and Compilers for Embedded Systems, 2015. 58–67

21 Purini S, Jain L. Finding good optimization sequences covering program space. ACM Trans Archit Code Opt, 2013, 9: 1–23

22 Park E, Cavazos J, Alvarez M A. Using graph-based program characterization for predictive modeling. In: Proceedings of the 10th International Symposium on Code Generation and Optimization, 2012. 196–206

23 Ashouri A H, Bignoli A, Palermo G, et al. MiCOMP: mitigating the compiler phase-ordering problem using optimization sub-sequences and machine learning. ACM Trans Archit Code Opt, 2017, 14: 1–28

24 Park E, Kartsaklis C, Cavazos J. HERCULES: strong patterns towards more intelligent predictive modeling. In: Proceedings of International Conference on Parallel Processing, 2014. 172–181

25 Martins L G A, Nobre R, Cardoso J M P, et al. Clustering-based selection for the exploration of compiler optimization sequences. ACM Trans Archit Code Opt, 2016, 13: 1–28

26 Ashouri A H, Mariani G, Palermo G, et al. COBAYN: compiler autotuning framework using bayesian networks. ACM Trans Archit Code Opt, 2016, 13: 1–25

27 Foleiss J H, Faustino d S A, Ruiz L B. The effect of combining compiler optimizations on code size. In: Proceedings of the 30th International Conference of the Chilean Computer Science Society, 2011. 187–194

28 Plotnikov D, Melnik D, Vardanyan M, et al. An automatic tool for tuning compiler optimizations. In: Proceedings of the 9th International Conference on Computer Science and Information Technologies Revised Selected Papers, Yerevan, 2014. 1–7

29 Lima E D D, Xavier T C D S, Silva A F D, et al. Compiling for performance and power efficiency. In: Proceedings of International Workshop on Power and Timing Modeling, Optimization and Simulation, 2013. 142–149

30 Narayanamurthy N, Pattabiraman K, Ripeanu M. Finding resilience-friendly compiler optimizations using metaheuristic search techniques. In: Proceedings of the 12th European Dependable Computing Conference (EDCC), 2016. 1–12

31 Sangchoolie B, Ayatolahi F, Johansson R, et al. A study of the impact of Bit-Flip errors on programs compiled with different optimization levels. In: Proceedings of Dependable Computing Conference, 2014. 146–157

32 Li F Q, Tang F L, Shen Y. Feature mining for machine learning based compilation optimization. In: Proceedings of the 8th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 2014. 207–214

33 Nazarian G, Strydis C, Gaydadjiev G. Compatibility study of compile-time optimizations for power and reliability. In: Proceedings of the 14th Euromicro Conference on Digital System Design, 2011. 809–813

34 Touati S A A, Barthou D. On the decidability of phase ordering problem in optimizing compilation. In: Proceedings of the 3rd Conference on Computing Frontiers, 2006. 147–156

35 Jantz M R, Kulkarni P A. Analyzing and addressing false interactions during compiler optimization phase ordering. Softw Pract Exper, 2014, 44: 643–679

36 Chebolu N A B S, Wankar R. A novel scheme for compiler optimization framework. In: Proceedings of International Conference on Advances in Computing, Communications and Informatics, 2015. 2374–2380

37 Nobre R, Martins L G A, Cardoso J M P. A graph-based iterative compiler pass selection and phase ordering approach. ACM SIGPLAN Not, 2016, 51: 21–30

38 Ashouri A H, Bignoli A, Palermo G, et al. Predictive modeling methodology for compiler phase-ordering. In: Proceedings of the 7th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and the 5th Workshop on Design Tools and Architectures For Multicore Embedded Computing Platforms, 2016. 7–12

39 Chen Y, Fang S D, Huang Y J, et al. Deconstructing iterative optimization. ACM Trans Archit Code Opt, 2012, 9:

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

1–30
40 Lu P J, Che Y G, Wang Z H. An effective iterative compilation search algorithm for high performance computing applications. In: Proceedings of IEEE International Conference on High PERFORMANCE Computing and Communications, 2008. 368–373
41 Xuan J F, Jiang H, Ren Z L, et al. Solving the large scale next release problem with a backbone-based multilevel algorithm. IEEE Trans Softw Eng, 2012, 38: 1195–1212
42 Ren Z L, Jiang H, Xuan J F, et al. Hyper-heuristics with low level parameter adaptation. Evol Computat, 2012, 20: 189–227
43 Chen X, Jiang H, Chen Z, et al. Automatic test report augmentation to assist crowdsourced testing. Front Comput Sci, 2019, 13: 943–959
44 Li X C, Jiang H, Kamei Y, et al. Bridging semantic gaps between natural languages and APIs with word embedding. IEEE Trans Softw Eng, 2018. doi: 10.1109/TSE.2018.2876006
45 Li X C, Jiang H, Liu D, et al. Unsupervised deep bug report summarization. In: Proceedings of IEEE/ACM International Conference on Program Comprehension, 2018. 144–155
46 Mou L L, Li G, Zhang L, et al. Convolutional neural networks over tree structures for programming language processing. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence, 2016. 1287–1293

Selection of compiler-optimization sequences

Guojun GAO[1], Zhilei REN[1*], Jingxuan ZHANG[2], Xiaochen LI[1] & He JIANG[1]

1. *School of Software, Dalian University of Technology, Dalian* 116024*, China*;
2. *College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing* 210016*, China*
* Corresponding author. E-mail: zren@dlut.edu.cn

Abstract In recent decades, compiler developers have designed and implemented many compiler-optimization options for a variety of complex optimization requirements; however, it is difficult to effectively adapt existing standard compiler-optimization sequences comprising several optimization options to complex compilation requirements. Because programs have different semantics and compilation goals, it is difficult to achieve desirable optimization results by directly employing standard compiler-optimization sequences. Additionally, the evolution of hardware architectures has increased the complexity of compilation environments, requiring compiler-optimization sequences to be adjusted accordingly. Therefore, selection of good optimization sequences for programs remains challenging. To address this, we reviewed 55 studies related to the selection of compiler-optimization sequences and summarized the research status of this area from several perspectives, including algorithmic approaches, research focuses, target compilers, and target benchmarks. The most popular algorithms used in this research area include heuristic search algorithms (e.g., genetic algorithms) and machine-learning algorithms (e.g., support vector machines). Over 80% of the existing studies focused on novel solutions or validation research. Furthermore, the most popular compiler and benchmark suites used for these experiments are GCC and miBench, respectively. This review offers insight into research trends associated with compiler optimization-sequence selection and provides possible directions for future studies in this field.

Keywords compiler, compiler-optimization sequence, heuristic search, machine learning

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050

Guojun GAO was born in 1992. He is a Ph.D. candidate at Dalian University of Technology. His main research interests include compiler testing and optimization.

Zhilei REN received his Ph.D. degree in computational mathematics from Dalian University of Technology, Dalian, China, in 2013. He is currently an associate professor at Dalian University of Technology. His current research interests include evolutionary computation, automatic algorithm configuration, and mining software repositories.

Jingxuan ZHANG was born in 1988. He received his Ph.D. in software engineering from Dalian University of Technology, Dalian, China, in 2018. He is currently a lecturer at Nanjing University of Aeronautics and Astronautics, Nanjing, China. His research interests include mining software repositories and API document analysis.

Xiaochen LI is a Ph.D. candidate at Dalian University of Technology. His research interests include mining software repositories, open-source software engineering, and software semantic analysis.

Downloaded to IP: 192.168.0.213 On: 2019-11-08 09:47:29 http://engine.scichina.com/doi/10.1360/N112019-00050