DOI:10.7544/issn1000-1239.2019.20180789
56(9):2012-2026,2019

学习模型指导的编译器优化顺序选择方法

刘 慧[1,2,3] 徐金龙[2] 赵荣彩[2] 姚金阳[2]

[1](河南师范大学计算机与信息工程学院 河南新乡 453007)

[2](数学工程与先进计算国家重点实验室(战略支援部队信息工程大学) 郑州 450002)

[3](河南省高校"计算智能与数据挖掘"工程技术研究中心(河南师范大学) 河南新乡 453007)

(liuhui806@126.com)

Compiler Optimization Sequence Selection Method Based on Learning Model

Liu Hui[1,2,3], Xu Jinlong[2], Zhao Rongcai[2], and Yao Jinyang[2]

[1](*College of Computer and Information Engineering, Henan Normal University, Xinxiang, Henan* 453007)

[2](*State Key Laboratory of Mathematical Engineering and Advanced Computing (Strategic Support Force Information Engineering University), Zhengzhou* 450002)

[3](*Engineering Technology Research Center for Computing Intelligence&Data Mining in Henan Province (Henan Normal University), Xinxiang, Henan* 453007)

Abstract For new applications and target platforms, it is often necessary to use the compiler for optimization sequence selection to improve the performance of target code. Iterative compilation enables the optimization sequence selection process automatically, with as many different versions of the program as possible within the allowable time and space range. However, iterative compilation method is a mechanical search that lacks the utilization of previous experience and requires large execution overheads. Therefore, an optimized compilation method is needed to automatically predict the performance of the transformed program without actually running. This paper presents Features ANN to select the optimization sequence of compiler. Features ANN is based on the supervised learning model. Firstly, the program feature is extracted by the program feature representation technique through a combination of dynamic and static feature. Then, the compiler optimization space is searched based on the program features, and it finds the best optimization of the current version of the program. Finally, training samples are formed by program features and optimal optimization, and an artificial neural network (ANN) is used to construct a learning model to predict the optimal optimization sequence of the new program. Experimental results show that, Features ANN can get the best performance compared with the existing iterative compilation and non-iterative compilation methods.

Key words compile optimization; optimization sequence; supervise learning; features extraction; artificial neural network (ANN)

摘 要 针对新的应用程序和目标平台通常需要使用编译器进行程序优化顺序选择,以提升目标代码性能.迭代编译可使优化顺序选择过程自动进行,在允许的时间空间范围内尽可能多地执行程序的不同版本,

收稿日期:2018-11-26;**修回日期**:2019-04-09

基金项目:国家重点研发计划项目(2016YFB0200503);河南师范大学青年科学基金项目(2015QK21);河南省高等学校重点科研项目(20A520018)

This work was supported by the National Key Research and Development Program of China (2016YFB0200503), the Youth Science Fund Project of Henan Normal University (2015QK21), and the Key Scientific Research Project of Henan Province Colleges (20A520018).

通信作者:徐金龙(xujinlong_pla@126.com)

但该方法是一种机械式搜索,缺少对先前获得经验的利用,需要较大的执行开销.因此,需要能自动预测变换后目标程序性能而不必实际运行程序的优化编译方法.提出一种编译器优化顺序选择方法:Features ANN.该方法首先采用动静结合的程序特征表示技术,对程序特征进行抽取;然后基于程序特征对编译优化空间进行搜索,找到当前程序版本的最佳优化;最后,由程序特征和最佳优化形成训练样本,基于人工神经网络(artificial neural network, ANN)形成监督学习模型,对新程序的最佳编译优化顺序进行预测.实验结果表明,Features ANN 与 2 种现有迭代编译方法 GraphDSE 和 ClusterDSE 比较时,在 2 种平台上相对于编译器标准优化级别-O3 分别获得 1.49x,1.25x,1.38x 和 1.41x,1.16x,1.22x 的执行时间加速比.此外,与现有非迭代编译方法相比时,Features ANN 也获得了有效的性能提升.

关键词　编译优化;优化顺序;监督学习;特征抽取;人工神经网络

中图法分类号　TP314

为应用程序选择最佳编译优化顺序是编译优化领域历久弥新的问题,而且是 NP 完全问题,编译器研究人员依靠他们对编译器后端的理解提出一些预定义的优化顺序.大型应用程序编译优化方案的提出需要研究人员花费很长的时间运行程序的不同优化版本,远不及体系结构和应用软件的更新速度.GCC(GNU compiler collection)编译器有超过 200 个的编译优化遍(pass),LLVM(low level virtual machine)编译器有超过 100 个的编译优化遍,并且这些优化在不同的应用层上工作,如分析遍、循环嵌套遍等.通常情况下,大多数编译优化是默认关闭的,编译器开发人员期望软件开发人员知道哪些优化对其代码是有益的,而软件开发人员可能并不熟悉编译优化相关知识.目前,大多商用编译器使用标准优化级别-O1,-O2,-O3 等为应用程序执行固定顺序的优化,对于大多数应用程序可以带来“平均良好性能”.然而,由于应用程序具有不同程序特征,为产生最佳性能提升,必须采用更有程序针对性的编译优化顺序[1].特别是针对嵌入式系统的编译器更加依赖于代码优化,因为使用的体系结构更加受限于内存大小、处理器速度等.如何为目标程序实施更有针对性的优化顺序,最大化程序性能仍然是一个亟需解决的关键问题.

在高性能计算领域中,并行计算机系统日益复杂.为了降低系统功耗,开发人员应用了不同的硬件和软件技术.目前主流的 P 级高性能计算机系统大多使用了 GPU[2] 和 MIC(many integrated core architecture)[3] 等众核处理器作为加速器或协处理器.在 2017 年 6 月的 TOP 榜单中,排名前十的机器中有 5 台使用了 GPU 或者 MIC 处理器作为加速器和协处理器[4].计算密集型应用程序将大部分的执行时间花费在适合于高级编译优化的循环嵌套中,例如稠密线性代数代码等.近年来提出的多面体编译优化技术亦致力于将数学表示集中在多面体模型的循环嵌套中[5-6].因此,复杂体系结构下为应用程序特定循环结构选择更有针对应的优化顺序,对程序性能提升至关重要.目前,通常使用迭代编译方法和基于机器学习的方法解决编译器优化顺序的选择问题.

迭代编译(iterative compilation)是一种针对通用程序的优化方法,该方法可有效集成不同的优化技术,适用于不同的体系结构[7].Chen 等人[8] 提出基于数据中心的迭代编译方法,通过在主服务器上收集性能统计数据,选择最佳编译优化及进行收益分析.Purini 等人[9] 使用下采样技术减少优化搜索空间,为应用程序选择最佳编译器优化顺序.Nobre 等人[10] 提出将编译器优化遍之间的变换用图形进行表示,然后通过图形采样的方式确定目标程序的优化顺序,能够有效减小搜索空间和提升收敛速度.Martins 等人[11] 基于遗传算法,采用聚类技术实施应用程序特定的优化顺序选择.在面向 FPGAs 的硬件编译环境下,Huang 等人[12] 分析了 2 种基于顺序插入的迭代方法,用于可变序列长度的编译优化顺序选择.采用迭代编译获取的程序性能加速明显好于静态编译,但在程序的迭代编译过程中会产生庞大的编译优化空间.此外,迭代编译是一种“无记忆”的优化搜索方法,不能利用已经获取的编译优化经验.

因此,编译优化人员提出将机器学习技术加入传统迭代编译优化过程,逐步形成基于机器学习的迭代编译方法[13-15].优化预测模型的输入为程序特征,输出为特定性能目标,如程序运行时间、优化选择和优化顺序等.理想情况下,预测模型独立于应用程序,能对不同程序版本进行快速评估,显著减少迭代编译代价[16].Agakov 等人[17] 提出 2 种优化顺序预测模型:独立同分布模型和马尔可夫模型,以提升

应用程序性能.预测模型在迭代编译过程中利用机器学习算法寻找更好的优化顺序,取得了一定的程序性能提升.但 Agakov 等人[17]提出的模型仅使用源代码程序特征,不能体现程序的当前版本优化状态,而程序的当前优化状态对接下来将使用的优化具有重要的影响.

Fursin 等人[18]提出基于 GCC 编译器的 Milepost GCC,将机器学习算法集成于 GCC 编译框架.Milepost GCC 基于源代码及程序中间表示抽取程序特征,统计不同指令和函数控制流图信息,使用机器学习技术自动调整程序优化变换.但 Milepost GCC 主要解决的是优化选择问题,基于固定的优化顺序从中选择能最有效提升代码性能的优化,而且其使用的是固定长度的特征向量.Cavazos 等人[19]提出一种基于机器学习的编译优化顺序预测方法,该方法使用性能计数器构建程序特征向量,并用于预测编译优化顺序.Dubach 等人[20]提出使用机器学习在嵌入式程序和微体系结构之间进行可移植的编译器优化.Hoste 等人[21]提出使用多目标进化算法为程序选择最佳编译优化.Kumar[22]提出通过在多核 CPU 上使用并行遗传算法选择最佳编译优化.Jantz 等人[23]采用修剪设计空间搜索技术,在较小的非联合优化子集上进行多阶段的优化顺序搜索.Ashouri 等人[24]提出优化顺序预测模型 COBAYN,基于贝叶斯网络将程序特征表示与编译优化遍相关联,以最大化程序性能.Park 等人[25]将基于多面体模型的迭代编译与机器学习相结合用以解决编译优化顺序问题,取得了较好的预测效果.现有基于机器学习的迭代编译方法大多数使用单独的静态程序特征或单独的动态程序特征进行优化预测模型的构建,如何采用更有效的程序特征表示方法需要进一步进行研究.此外,为了进一步降低编译器优化顺序选择过程中的迭代编译开销,如何构建更有效的优化预测模型,也亟需进行更深入的研究.

人工神经网络(artificial neural network, ANN)是一种模拟人脑功能对数据信息进行加工处理的系统化方法.本文提出基于机器学习的编译器优化顺序选择方法 Features ANN,通过 ANN 基于监督学习模型进行编译器优化顺序选择.主要贡献有 3 个方面:

1) 使用动静结合的程序特征表示方法,通过主成分分析技术将高维程序特征降维为低维程序特征,在不降低程序表示性的基础上,为优化顺序预测模型选择最佳程序特征表示.

2) 优化顺序选择模型 Features ANN 的样本为由降维后的程序特征和当前最佳优化构成的二元组,利用人工神经网络建立统计模型并集成于编译器框架,为目标程序选择当前优化状态下的最佳优化及进行编译过程驱动.面对新的应用程序,Features ANN 将新程序特征作为其输入,并预测最佳优化,从而生成新程序的最佳优化顺序.

3) 在 2 种平台上采用动静结合特征作为 Features ANN 预测模型输入时,相对于 GCC 8.1.0 编译器标准优化级别-O3 分别获得 1.49x 和 1.41x 的程序执行时间加速.此外,与现有迭代编译方法和非迭代编译方法相比时,均获得了最佳的程序性能提升.

1 学习模型指导的优化顺序选择模型

1.1 优化顺序问题的形式化描述

为了对优化顺序选择问题有更直观的认识,我们首先对编译优化的选择空间进行形式化描述.定义布尔向量 $\boldsymbol{o}$,元素 o_i 表示不同的编译优化,每个优化可以被启用($o_i=1$)或禁用($o_i=0$).编译优化的选择空间属于 n 维布尔空间,由向量空间 $\boldsymbol{o}_{\text{selection}}$ 表示为

$$\boldsymbol{o}_{\text{selection}}=(0,1)^n. \tag{1}$$

当对应用程序 a_i 进行优化时,n 表示编译优化总数,由一个指数空间作为其上限.例如,当 $n=10$ 时,目标应用程序 a_i 有 1024 种可选择的优化方式.当应用程序有 N 个不同的优化版本 $A=\{a_0,a_1,\cdots,a_n\}$时,将产生更大的优化空间.

当使用固定优化序列长度且不重复使用优化时,定义编译优化顺序为 n 维向量空间为 $o_{\text{phase}}=n!$,其中,n 表示编译优化总数.当使用动态优化序列长度且可重复使用优化时,优化顺序空间扩展为

$$|\boldsymbol{o}_{\text{phase\_repetition}}|=\sum_{i=0}^{m}n^i, \tag{2}$$

其中,m 表示优化序列最大期望长度.假设 n 和 m 均为 10,将产生高达 110 亿种不同的编译优化顺序.当优化序列最大期望长度 m 没有固定值时,编译优化顺序问题没有上界.

1.2 监督学习模型指导的优化顺序选择模型

监督学习模型指导的编译器优化顺序预测模型框架如图 1 所示,包括数据收集、模型训练和模型预测阶段.在数据采集阶段,首先根据程序特征表示方法,使用设计空间探索引擎(design space exploration, DSE)抽取程序特征$\{F_1,F_2,\cdots,F_n\}$,提取目标程序

特征值$\{f_1,f_2,\cdots,f_n\}$.然后,使用 DSE 将优化集合$\{o_1,o_2,\cdots,o_m\}$中的优化 o_i 分别应用于目标程序.DSE 通过启用和禁用优化实施不同的优化、优化顺序及测试应用程序运行时间.由于程序在每应用一次优化后,特征值将会发生相应的改变,需要根据更新后的特征值确定下一阶段应使用的优化.因此,我们设定 DSE 在每应用一次优化后重新提取特征值$\{f_1,f_2,\cdots,f_n\}$,以表示程序的当前优化状态.并根据程序的当前优化状态再次选择优化集合$\{o_1,o_2,\cdots,o_m\}$中的优化 o_i 和测试运行时间.数据收集阶段收集的训练数据由$\{o_i,f_1,f_2,\cdots,f_n,s_i\}$构成.其中,$o_i$ 表示优化变换,$\{f_1,f_2,\cdots,f_n\}$表示程序特征值,s_i 表示特征值为$\{f_1,f_2,\cdots,f_n\}$时程序实施优化 o_i 可获得的加速比.当产生最大加速比 $s_{\max}$时,对应的优化 o_i 即为程序当前状态下可使用的最佳优化.

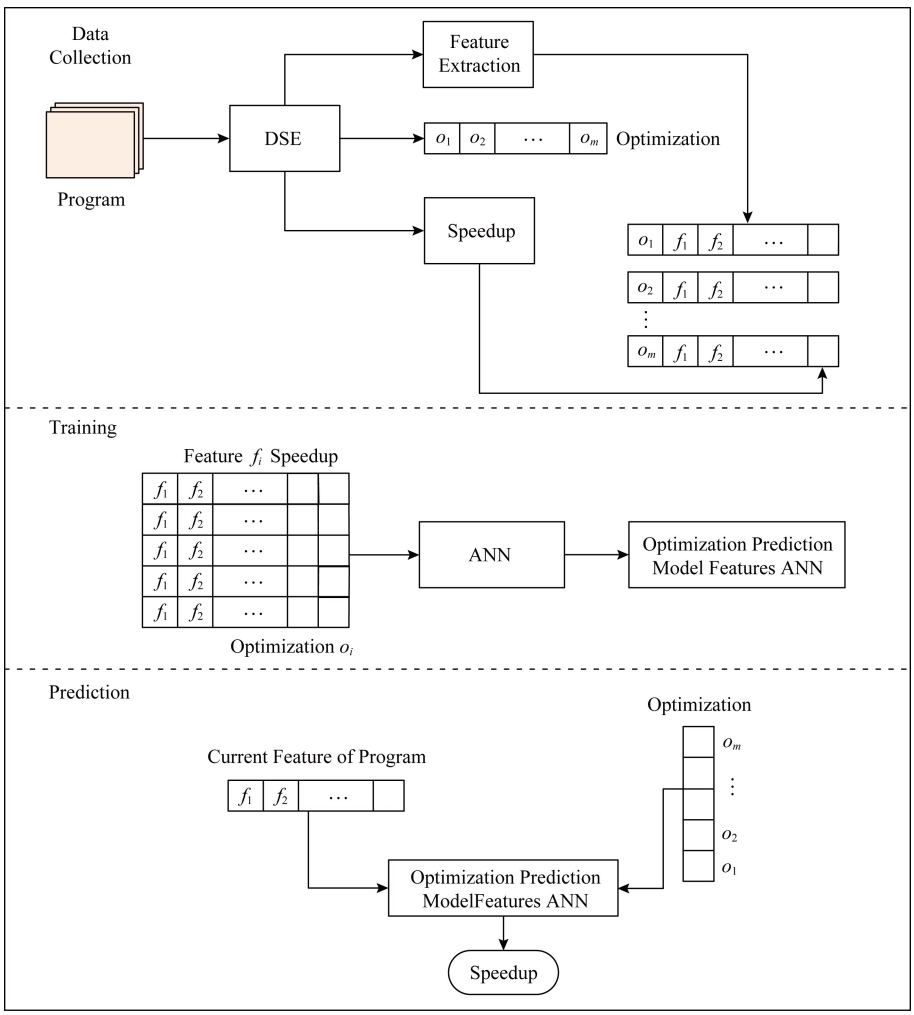


Fig. 1　Compile optimization sequence prediction model

图 1　编译优化顺序预测模型

在模型训练阶段,基于收集到的训练数据,采用 ANN 进行模型训练,构建程序优化及优化顺序预测模型 Features ANN.在模型预测即使用阶段,基于程序在当前优化状态下的特征值,通过存储在预测模型中的知识为新程序预测当前状态下应用不同优化时的加速比,实现最大加速比对应的优化即为程序在当前状态下应使用的最佳优化,从而生成新程序的最佳优化顺序.优化终止条件为达到预先设定的最大优化序列长度,或连续为程序当前状态预测相同的优化.

2　程序特征表示

2.1　程序特征

现有程序特征表示方法包括:

1) 静态程序特征表示.静态特征通常在语法

分析过程中、编译过程中提取.

2) 动态程序特征表示.动态特征在程序运行过程中提取,通常具有更好的程序表示性,但为获取动态特征往往需要反复执行程序的运行.

Features ANN 优化顺序预测模型使用动静结合的特征表示方法(dynamic static features combination, DSFC),为预测模型提供更细粒度的输入.静态和动态特征列表如表 1 和表 2 所示[14].

Table 1 List of Static Features

表 1 静态特征列表

Serial Number	Program Feature
1	Number of basic blocks in the function
2	Number of basic blocks with a single successor
3	Number of basic blocks with two successors
4	Number of basic blocks with more than two successors
5	Number of basic blocks with a single predecessor
6	Number of basic blocks with two predecessors
7	Number of basic blocks with more than two predecessors
8	Number of basic blocks with a single predecessor and successor
9	Number of basic blocks with a single predecessor and two successors
10	Number of basic blocks with two predecessors and one successor
11	Number of basic blocks with two successors and two predecessors
12	Number of basic blocks with more than two successors and more than two predecessors
13	Number of instructions less than 15
14	Number of instructions in [15, 500]
15	Number of basic blocks with instructions greater than 500
16	Number of edges in the control flow graph
17	Number of critical edges in the control flow graph
18	Number of abnormal edges in the control flow graph
19	Number of direct calls in the function
20	Number of conditional branches in the function
21	Number of assignment instructions in the function
22	Number of binary integer operations in the function
23	Number of binary floating point operations in function
24	Number of instructions in the function
25	Average of number of instructions in basic blocks
26	Average number of phi-nodes at the beginning of a block
27	Average of arguments for a phi-node
28	Number of basic blocks with no phi nodes

Table 2 List of Dynamic Features

表 2 动态特征列表

Category	Performance Counter Selection
Cache Line Access	CA-CLN, CA-ITV, CA-SHR
Level 1 Cache	L1-DCA, L1-DCH, L1-DCM, L1-ICA, L1-ICH, L1-ICM, L1-LDM, L1-STM, L1-TCA, L1-TCM
Level 2 & 3 Cache	L2-DCA, L2-DCM, L2-DCR, L2-DCW, L2-ICA, L2-ICH, L2-ICM, L2-LDM, L2-STM, L2-TCH, L2-TCR, L2-TCW, L2/L3-TCA, L2/L3-TCM
Branch Related	BR-CN, BR-INS, BR-MSP, BR-NTK,BR-PRC
Interrupt/Stall	HW-INT, RES-STL
TLB	TLB-DM, TLB-IM, TLB-SD, TLB-TL
Total Cycle/Instructions	TOT-CYC, TOT-IIS, TOT-INS
Load/Store Instructions	LD-INS, SR-INS
SIMD Instructions	VEC-DP, VEC-INS, VEC-SP

2.2　特征降维技术

程序特征的选择是构建优化顺序预测模型的关键,本文采用主成分分析法(principal component analysis, PCA)为目标程序选择最主要的静态特征和动态特征.当数据空间维度大于3维时,将无法在空间坐标系下用图形(如点、线、面)表示样本点.程序特征高维空间的多变量数据不能以图形化方式直观展示其空间特征分布规律,如样本相似度、样本中的异常点等信息.PCA的核心思想是通过正交变换,将一个变量维数较高而且变量之间存在相互关联的数据矩阵映射到一个较低维数的主成分空间.程序特征的PCA计算过程为:

Step1. 假定程序特征中有 p 个样本,每个样本共有 n 个变量,样本数据可构成一个 $p\times n$ 阶矩阵:

$$\boldsymbol{X}=\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1i} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2i} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots & & \vdots \\ x_{p1} & x_{p2} & \cdots & x_{pi} & \cdots & x_{pn} \end{pmatrix}_{p\times n}. \quad (3)$$

对样本矩阵进行标准化变换,得到标准化阵 $\boldsymbol{Z}$:

$$Z_{ij}=\frac{x_{ij}-\bar{x}_j}{s_j}, i=1,2,\cdots,p, j=1,2,\cdots,n, \quad (4)$$

其中,

$$\bar{x}_j=\frac{1}{p}\sum_{i=1}^{p}x_{ij}, \quad (5)$$

$$s_j^2=\frac{1}{p-1}\sum_{i=1}^{p}(x_{ij}-\bar{x}_j)^2. \quad (6)$$

Step2. 对标准化阵 $\boldsymbol{Z}$ 求相关系数矩阵:

$$\boldsymbol{R}=(r_{ij})_{n\times n}=\frac{\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}}{p-1}. \quad (7)$$

Step3. 解样本相关阵 $\boldsymbol{R}$ 的特征方程 $|\lambda\boldsymbol{I}-\boldsymbol{R}|=0$,求出 n 个特征根,按照特征值的大小倒序排列.对于每一个特征值 λ_i 求出其所对应的特征向量 $\boldsymbol{e}_i$ $(i=1,2,\cdots,n)$.

Step4. 主成分的选取规则:

$$I_i=\frac{\lambda_i}{\sum_{k=1}^{i}\lambda_k}(i=1,2,\cdots,n), \quad (8)$$

$$S_i=\frac{\sum_{k=1}^{i}\lambda_k}{\sum_{k=1}^{n}\lambda_k}(i=1,2,\cdots,n), \quad (9)$$

其中,统计量 I_i 代表某一主成分 λ_i 的贡献率,统计量 S_i 代表某一主成分 λ_i 的累积贡献率.k 的选择原则是取累计贡献率90%,保证主成分变量包括原始数据的大多数信息,即求得 $S_i\approx 0.9$ 的所有 λ_i 所对应的主成分 f_i.

Step5. 载荷系数的计算.观测值 $x_i(i=1,2,\cdots,n)$ 在主成分 f_i 上的得分为

$$I_{ij}=\sqrt{\lambda_i}e_{ij}(i=1,2,\cdots,n;j=1,2,\cdots,m). \quad (10)$$

Step6. 根据程序特征,对程序特征主成分进行命名,对主成分及其荷载进行相关解释.

主成分提取原则一般选取主成分累计贡献率超过90%的前 m 个主成分,经过实验分析表明,由于程序特征值的冗余和协方差,程序特征可以降维至5维特征向量,此时仍能保持99%的数据差异性.

3　ANN构建

3.1　基于ANN的优化顺序预测

在编译器中确定合适的优化顺序是一个十分复杂的问题,我们可以将优化顺序问题定义为马尔可夫过程(Markov process, MP),即基于当前程序版本状态做出接下来应用何种优化的决定.本文以ANN为基础构造程序变换优化顺序预测模型Features ANN.Features ANN的基本思想是持续询问ANN预计使用何种优化可为当前状态下的程序产生最佳性能.ANN使用当前状态下的代码特征作为输入,利用网络层次表示特征之间复杂的非线性关系,并与优化过程中特定点的最佳优化相关联.Features ANN模型框架如图2所示:

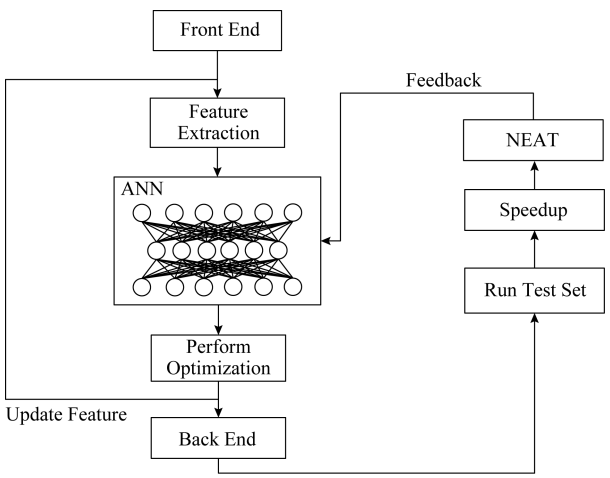


Fig. 2　Process of Features ANN compile optimization selection

图2　Features ANN编译优化选择过程

在模型训练阶段,采用扩张拓扑的神经进化算法(neuro-evolution for augmenting topologies, NEAT)生成用来控制优化顺序的ANN,通过对每个函数

应用不同的优化,并记录程序运行时间进行 ANN 评估.Features ANN 迭代输出优化 o_i 的过程如图 3 所示.ANN 的输入为当前状态下的代码特征 $\{f_1, f_2, \cdots, f_n\}$,输出为应用不同优化 o_i 后的收益概率 P,选择最大概率 P 对应的优化 o_{best} 作为当前状态下应使用的优化.应用优化 o_{best} 后将形成一个新的程序版本,特征值发生改变,将更新后的特征值重新输入 Features ANN,使其输出下一个要应用的优化 o_{i+1}.当 ANN 达到最大代数或连续若干代性能没有变化时,停止使用优化,生成目标程序优化顺序 $\{o_1, o_2, \cdots, o_m\}$.模型预测阶段将 ANN 集成于编译器,并作为新程序优化顺序的启发式规则进行使用.Features ANN 优化顺序选择算法如算法 1 所示.

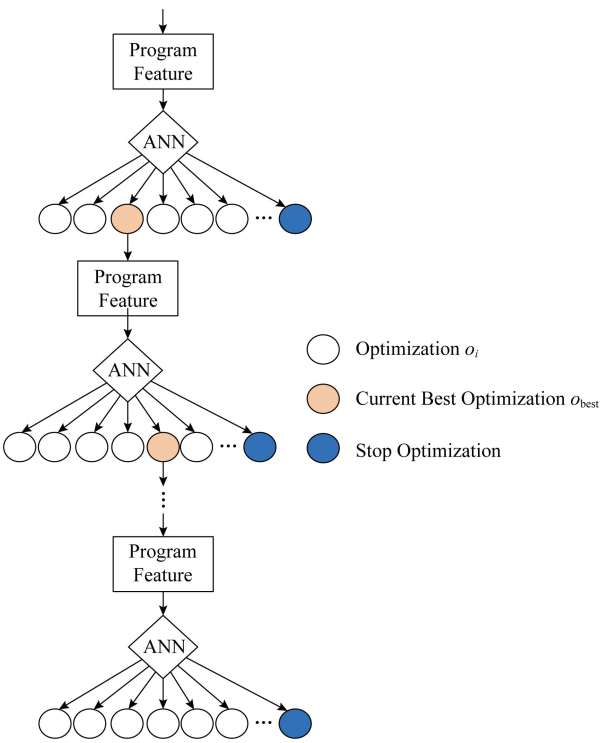


Fig. 3 Process of Features ANN output optimization o_i iteratively

图 3 Features ANN 迭代输出优化 o_i 过程

算法 1. 基于 ANN 的优化顺序选择算法.

输入:程序特征值 $f_1, f_2, \cdots, f_n$,优化 $\{o_1, o_2, \cdots, o_m\}$;

输出:最佳优化序 $o^* = \{o_1, o_2, \cdots, o_m\}$ 及其对应的程序运行时间 T^*.

Step1. 初始化:设置每代神经网络个数 N、网络代数 M,并随机生成第 1 代网络.

Step2. 输入程序 C 当前状态特征值 $f_1, f_2, \cdots, f_n$ 作为当前代网络输入层神经元的输入.

Step3. 计算每个网络的适应度 $Fitness(T)$,选择前 10 个具有最大适应度值的网络传播到下一代产生新的网络

$$Fitness(T) = \frac{1}{Runtime(f)}.$$

Step4. 交叉和变异产生新一代网络.变异包括在网络隐藏层现有边上增加一个神经元(概率设置为 0.1%)、增加一个新边(概率设置为 0.5%)或删除一个现有边(概率设置为 0.9%).

Step5. 当网络到达最大代数 M 或连续若干代性能没有变化,将具有最大适应度的优化作为当前程序版本状态下的最佳优化 o_{best}.

Step6. 算法终止条件:当达到最大优化个数或重复使用优化没有性能提升时,算法终止,转至 Step7,否则更新函数特征,转至 Step2.

Step7. 输出最佳优化序 $o^* = \{o_1, o_2, \cdots, o_m\}$ 及其对应的函数执行时间 T^*.

3.2 NEAT

由于优化顺序预测模型 Features ANN 处于动态编译环境中,因此 ANN 及特征抽取过程的开销要小,否则应用该网络进行优化顺序选择的代价会高于调整优化顺序带来的收益.本文采用 NEAT 算法[26]为程序变换优化顺序选择模型构建 ANN.NEAT 算法开始于若干没有隐结点的最小网络,在进化的实施过程中通过变异引入新结构,从而进行网络扩张.并通过适应度评估网络优劣,淘汰适应度较低的个体.由于种群从最小结构开始,优化搜索空间的维度也将是最小的.因此,NEAT 算法总是搜索比其他进化神经网络算法更低的维度空间,网络构建开销小于其他同类算法.

NEAT 进化神经网络的过程是从只有输入输出结点的简单网络开始逐步增加网络复杂程度,具有较大的结点连接自由度.如图 4 所示,NEAT 采用结点基因编码进行网络结构和连接权值的描述.当创建新的结点和连接时,借助于所产生的历史标记,从而避免竞争约定问题.NEAT 尝试将构成的网络尺寸最小化,在进化开始时让群体中每个神经网络都具有最小的拓扑结构,在进化过程中始终保持在网络中随机逐个添加神经元和连接.该过程和自然界生物体的生长过程一样,随着时间的推移,不断进化、不断增加复杂性,从而试图构建最小化的最优网络.

NEAT 算法的适应度值为训练集中测试集性能的几何平均值:

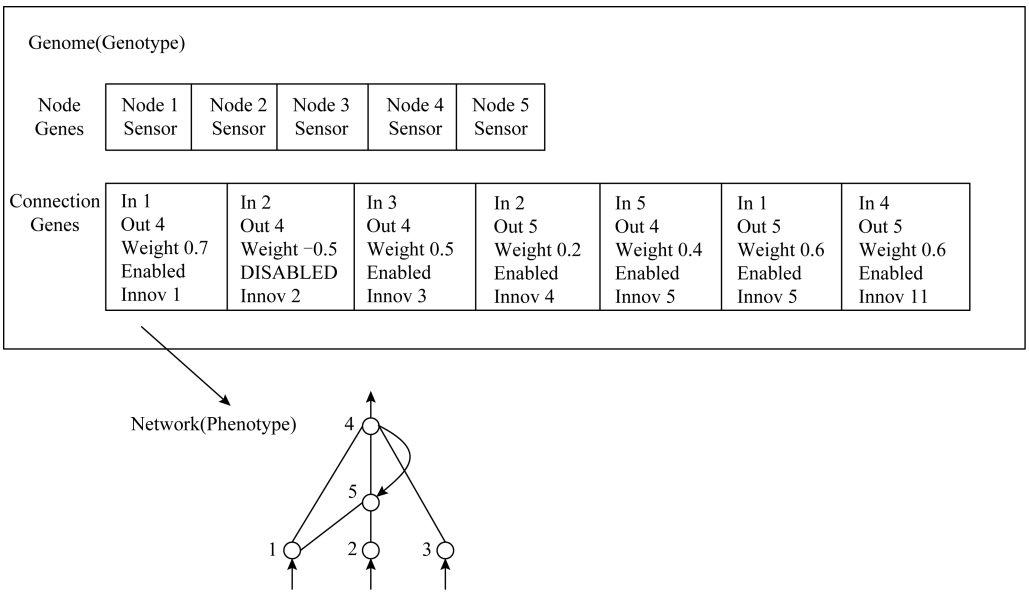


Fig. 4　NEAT gene mapping

图 4　NEAT 基因映射

$$Fitness(S)=\frac{\sum_{s\in S}Speedup(s)}{|S|},\tag{11}$$

其中,$|S|$ 表示训练集个数,$Speedup(s)$表示程序运行时间加速比:

$$Speedup(s)=\frac{Runtime(s_{def})}{Runtime(s)},\tag{12}$$

其中,$Runtime(s_{def})$表示测试集 s 使用默认优化顺序时的运行时间,$Runtime(s)$表示使用预测模型预测的优化顺序时测试集 s 的运行时间.

4　实　　验

我们在 2 种平台上进行 Features ANN 优化顺序预测模型的训练和评估,模型输入为程序热点函数特征,输出为相应的最佳优化顺序.

1) 平台 1. 实验编译环境为 Linux 操作系统,版本为 Readhat Enterprise AS 5.0,实验平台为国产处理器申威 26 010,CPU 主频为 2.5 GHz,L1 数据 cache 为 32 KB,L2 cache 为 256 KB,向量寄存器宽度为 256 b,可同时处理 4 个浮点型数据或 8 个整型数据.

2) 平台 2.实验编译环境为 Linux 操作系统,版本为 Readhat Enterprise AS 5.0,实验平台为 Intel Xeon E5520 处理器,CPU 主频为 2.26 GHz,L1 数据 cache 为 32 KB,L2 cache 为 1 MB,二级 Smart cache 8 MB.

实验中训练集测试用例为 SPEC CPU2006 测试集中的 2 000 个热点循环,测试集用例为 NPB 测试集和表 3 所示大型科学计算程序中的热点函数.实验将标准优化级别-O2 及图 5 所示优化作为优化空间,将新的优化顺序与标准优化级别-O3 的默认优化顺序进行性能对比,优化的启用/禁用通过使用 GCC8.1.0 编译优化选项完成.对程序热点函数特征采用动静结合特征表示方法 DSFC,将抽取的特征作为 PCA 过程的输入.我们分别比较了采用 Features ANN 和现有迭代编译方法、非迭代编译方法时测试集用例的程序性能.当现有方法使用的优化和测试集与本文不同时,我们采用本文的优化和测试集重新进行实验.对抽取的每个热点函数采用添加时间戳的方式进行运行时间计时,为减少噪声影响,每个样本运行 10 次,取执行时间平均值.

Table 3　Large Scientific Calculation Programs

表 3　大型科学计算程序

Program	Description
SWE	Explicit solution of the global shallow atmospheric water wave equation
OpenCFD	Fluid mechanics
FDM	The forward modeling program of 3D seismic wave equation
WRF	Weather forecast
GKUA	The solution of complex hypersonic flow around each watershed

-fauto-inc-dec	-falign-jumps	-ftree-tail-merge	-fstrict-overflow	-fcombine-stack-adjustments
-fcompare-elim	-fcprop-registers	-ftree-forwprop	-ftree-pre	-finline-functions-called-once
-ftree-dce	-fcaller-saves	-fgcse-after-reload	-ftree-vrp	-fmove-loop-invariants
-fipa-profile	-fif-conversion	-fssa-backprop	-funswitch-loops	-frerun-cse-after-loop
-ftree-ccp	-fif-conversion2	-fstrict-aliasing	-fpartial-inlining	-fdce -fdefer-pop
-fpeel-loops	-fsplit-paths	-fbranch-count-reg	-fssa-phiopt	-ftree-dominator-opts
-fshrink-wrap	-ftree-phiprop	-ftree-partial-pre	-finline-functions	-fguess-branch-probability
-ftree-sink	-ftree-slsr	-fipa-cp-clone	-fschedule-insns2	-ftree-loop-distribute-patterns
-fthread-jumps	-falign-functions	-fipa-bit-cp	-fsplit-wide-types	-fisolate-erroneous-paths-dereference
-falign-loops	-falign-labels	-findirect-inlining	-ftree-coalesce-vars	-foptimize-sibling-calls
-fcrossjumping	-ftree-vectorize	-ftree-bit-ccp	-foptimize-strlen	-freorder-blocks-algorithm=stc
-ftree-pta	-fipa-pure-const	-ftree-copy-prop	-fschedule-insns	-freorder-blocks-and-partition
-ftree-ter	-fdevirtualize	-funit-at-a-time	-fsched-interblock	-fdelete-null-pointer-checks
-ftree-sra	-fipa-reference	-fcse-follow-jumps	-freorder-functions	-fdevirtualize-speculatively
-ftree-fre	-freorder-blocks	-fcse-skip-blocks	-fhoist-adjacent-loads	-fexpensive-optimizations
-ftree-dse	-fpeephole2	-fipa-cp-alignment	-fpredictive-commoning	-ftree-loop-distribution
-fgcse	-flra-remat	-fforward-propagate	-fdelayed-branch -fdse	-finline-small-functions
-fgcse-lm	-fsched-spec	-fmerge-constants	-ftree-switch-conversion	-ftree-builtin-call-dce
-fipa-cp	-fipa-sra	-fipa-icf	-fcode-hoisting	-ffast-math

Fig. 5 List of Optimization(GCC8.1.0)

图 5 优化列表(GCC8.1.0)

本文实验主要从 4 个方面进行:1)Features ANN 采用不同特征时的性能比较.分别采用静态程序特征表示、动态程序特征表示和动静结合程序特征表示作为优化顺序预测模型 Features ANN 的输入,对比分析模型的预测性能.2)Features ANN 与迭代编译方法的性能比较.比较测试集程序采用 Features ANN 与 2 种现有迭代编译方法时的程序性能.3)Features ANN 与非迭代编译方法的性能比较.比较测试集程序采用 Features ANN 和 3 种现有非迭代编译方法时的程序性能.4)离线学习和在线预测时间开销比较.对比分析 Features ANN 和现有迭代编译、非迭代编译方法在训练数据收集、模型构建、特征抽取和模型预测上的时间开销.

5 实验结果分析

5.1 采用不同特征时的预测模型性能

基于现有程序特征表示技术,我们对比分析 3 类程序特征表示方法对优化顺序预测模型的影响:静态程序特征表示、动态程序特征表示和动静结合程序特征表示.其中,静态程序特征采用 Milepost GCC 静态程序,包括以固定长度特征向量表示的静态源代码及中间表示特征[18].动态程序特征采用基于性能计数器的动态特征表示(performance counter,PC),PC 能实时采集、分析系统内的应用程序、服务等性能数据,以此来分析系统瓶颈、监视组件表现[19].动静结合特征表示我们的程序特征表示方法 DSFC.

Features ANN 分别采用不同特征表示方法,为 NPB 测试集和表 3 所示科学计算程序在平台 1 和平台 2 上预测优化顺序时,预测模型性能如图 6 和图 7 所示.其中,Milepost GCC 表示静态程序特征,PC 表示动态程序特征,DSFC 表示本文提出的动静结合程序特征表示.Speedup 表示相对于 GCC 8.1.0 标准优化级别-O3 产生的性能加速比.

实验结果表明,平台 1 上基于 Milepost GCC 静态程序特征、PC 动态特征和 DSFC 动静结合特征为 NPB 测试集和大型科学计算程序预测优化顺序时,Features ANN 相对于 GCC 标准优化级别-O3 产生的平均加速比分别为 1.19,1.23,1.49.平台 2 上产生的平均加速比分别为 1.16,1.20,1.41.总体来说,对当前测试用例而言,采用 DSFC 动静结合特征可以产生最好的预测性能,PC 动态特征次之,Milepost GCC 静态特征具有相对较低的预测性能.

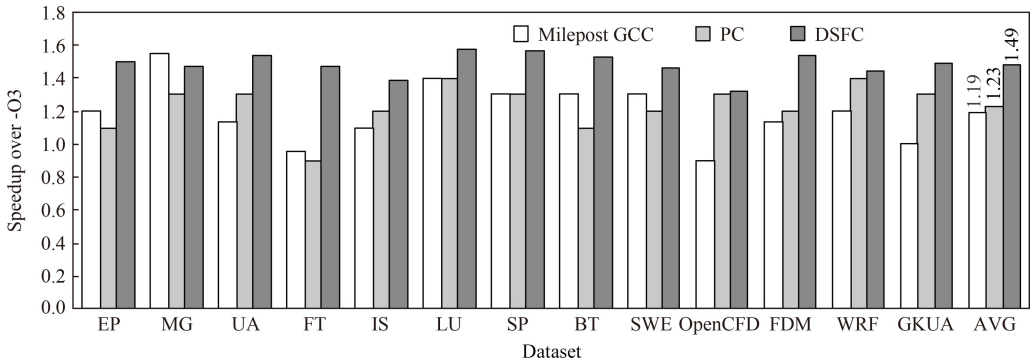


Fig. 6　Speedup on platform 1 when adapting different features

图 6　采用不同特征时平台 1 上测试程序加速比

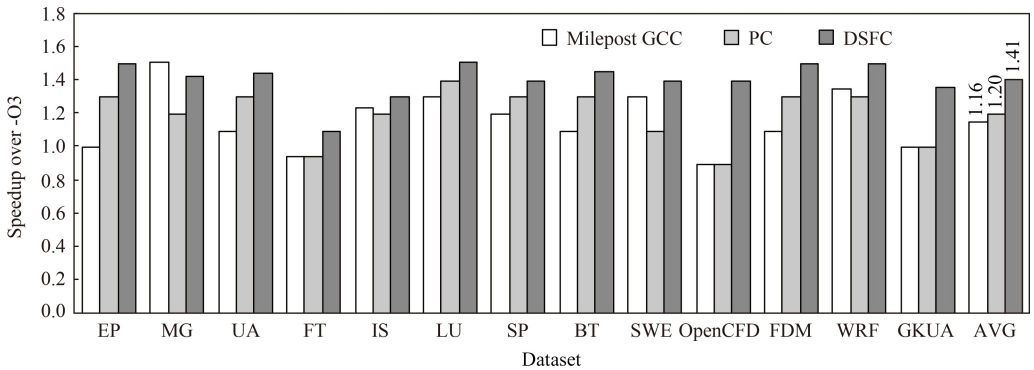


Fig. 7　Speedup on platform 2 when adapting different features

图 7　采用不同特征时平台 2 上测试程序加速比

经程序分析可知,测试集 LU 的主要热点函数为 rhs,占程序执行时间的 24.98%.采用 DSFC 动静结合特征时,Features ANN 为 rhs 预测的优化顺序为{-O2,-fpeel-loops,-funroll-loops,-fprefetch-loop-arrays,-fgcse-after-reload,-ffast-math}.采用 Milepost GCC 静态特征时,Features ANN 为 rhs 预测的优化顺序为{-O2,-fpeel-loops,-funroll-loops,-fprefetch-loop-arrays,-fgcse-after-reload,-ftree-vectorize,-ffast-math}.采用 PC 动态特征时,Features ANN 为 rhs 预测优化顺序为{-O2,-fpeel-loops,-fprefetch-loop-arrays,-funroll-loops,-ffast-math}. 采用 GCC-O3 默认优化、DSFC 动静结合特征、Milepost GCC 静态程序特征和 PC 动态特征时对应的函数执行时间分别为:45.975 s,37.364 s,41.864 s,40.376 s.采用 DSFC 动静结合特征预测的优化顺序除使用标准优化级别-O2 外,首先对该热点循环进行循环剥离,然后进行循环展开,循环展开可以将迭代次数少的循环进行展开以充分利用寄存器.循环中含有大量的 3 维数组和 4 维数组引用,使用-fprefetch-loop-arrays 可以对数据预取,对含有大数组的程序可以产生较好的性能提升.此外,由于循环中含有大量的变量引用,在计算时会被重复使用,使用-fgcse-after-reload 优化可以消除重复性的访存.采用 Milepost GCC 静态特征预测的优化顺序会对 rhs 函数进行向量化处理,但该程序含有间接访存,且语句间含有阻止向量化的依赖环,内层循环迭代次数小,循环多为不完美循环嵌套,若对其进行向量化,将会引入一些数据整理指令,增大开销,性能反而会下降.采用 PC 动态特征预测的优化顺序在数据预取之后进行循环展开,会增加预取次数,且预测的优化中没有使用-fgcse-after-reload,获得的程序执行时间加速比小于采用 DSFC 动静结合特征产生的加速比.

DSFC 动静结合特征表示优于 Milepost GCC 静态特征表示,原因在于 Milepost GCC 特征主要包括基本块和边的信息,这些信息是以整个程序为单位的固定长度的特征表示,而 DSFC 采用动静结合特征表示方法,而且程序特征是采用 PCA 方法选择的,不是固定不变的,更能表示出不同程序的主要信息.但是有一些程序如 MG 采用 Milepost GCC 特征时性能优于 DSFC,原因是 Milepost GCC 有一些

静态特征没有包括在在基于 DSFC 的特征中,这可能对特定程序有影响.DSFC 动静结合特征表示也优于单独使用 PC 动态特征表示获得的性能.因此,在 Features ANN 预测模型中使用动静结合程序特征 DSFC 有利于进一步提升模型的优化顺序预测性能.

5.2 Features ANN 与迭代编译方法的比较

为将 Features ANN 与现有迭代编译方法进行比较,我们选择与 Nobre[10] 和 Martins[11] 的研究进行对比分析.Nobre 等人[10] 提出一种基于迭代编译的优化顺序搜索方法,该方法首先将编译优化变换用图形方式进行表示,然后通过图形采样确定程序的优化顺序,能够有效减小优化搜索空间和提升收敛速度.Martins 等人[11] 基于聚类方法,采用遗传算法指导的设计空间探索技术选择应用特定的优化顺序,该方法是最近的基于迭代编译方法进行优化顺序搜索的典型代表.采用 Features ANN 和现有迭代编译方法为 NPB 测试集、大型科学计算程序在平台 1 和平台 2 上预测优化顺序时,预测模型性能如图 8 和图 9 所示.其中,GraphDSE 表示 Nobre 等人[10] 提出的基于图形的优化顺序预测模型,ClusterDSE 表示 Martins 等人[11] 提出的基于聚类的预测模型,Features ANN 表示本文提出的预测模型.

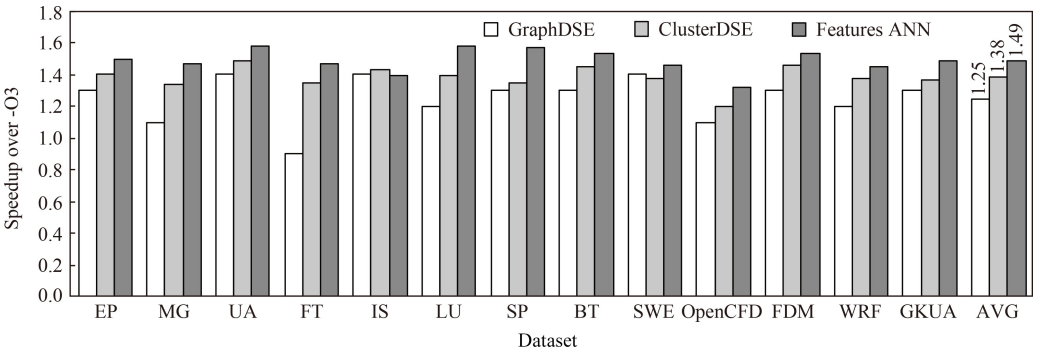


Fig. 8 Comparison with iterative compilation (platform 1)

图 8 Features ANN 与迭代编译方法的性能比较(平台 1)

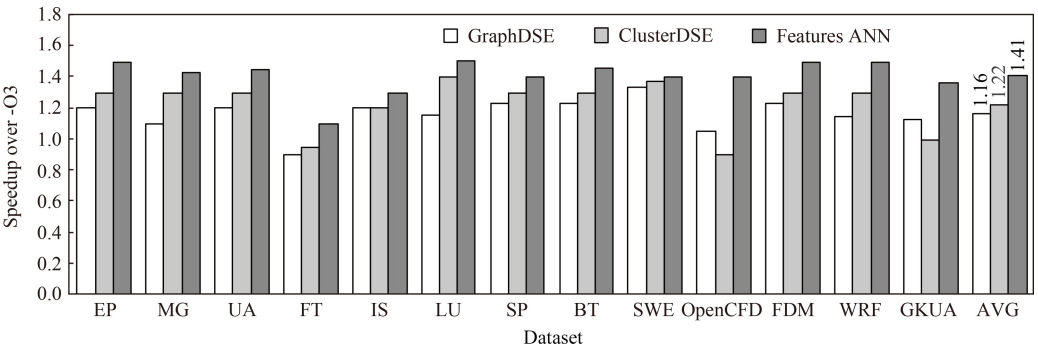


Fig. 9 Comparison with iterative compilation (platform 2)

图 9 Features ANN 与迭代编译方法的性能比较(平台 2)

从图 8 可以看出,平台 1 上基于迭代编译方法为测试集预测优化顺序时,Features ANN,GraphDSE,ClusterDSE 相对于 GCC 标准优化级别-O3 的平均加速比分别为 1.49,1.25,1.38.对当前测试用例而言,Features ANN 具有最好的预测性能,ClusterDSE 次之,GraphDSE 具有相对较低的预测性能.经程序分析可知,测试集 UA 的主要热点函数为 laplacian,占程序执行时间的 40%.Features ANN 为 laplacian 函数预测的优化顺序为{O2,-fpredictive-commoning,-funswitch-loops,-fpeel-loops,-ftree-loop-distribution,-ftree-vectorize,-ffast-math}.GraphDSE 为 laplacian 预测的优化顺序为{-O2,-fpredictive-commoning,-funswitch-loops,-ftree-loop-distribution,-ftree-vectorize,-fpeel-loops}. ClusterDSE 为 laplacian 预测的优化顺序为{-O2,-fpredictive-commoning,-funswitch-loops,-fpeel-loop,-ftree-loop-distribution,-ftree-vectorize}.采用 GCC-O3 默认优化、Features ANN、GraphDSE、ClusterDSE 时对应

的函数执行时间分别为 43.87 s,38.57 s,43.19 s,42.50 s.由于函数 laplacian 中含有多个完美嵌套循环,且不含阻止向量化的依赖环,适合做向量化处理.Features ANN 预测的优化顺序除使用标准优化级别-O2 外,在为函数进行向量化处理前使用预测常见优化、分支外提,循环剥离与分布等优化.在进行向量化处理后调用快速数学库优化,打开-ffast-math,实验表明在精度要求不苛刻的情况下使用这种激进的优化方式可以获得较大的性能提升.而 ClusterDSE 预测的优化顺序没有使用-ffast-math,程序性能略差于 Features ANN.GraphDSE 预测的优化顺序在优化选择上与 ClusterDSE 相同,但优化顺序不同,即在向量化后进行循环剥离,从而导致一些可以使用循环剥离的循环体失去了向量化的机会.

从图 9 可以看出,平台 2 Features ANN,GraphDSE,ClusterDSE 相对于 GCC 标准优化级别-O3 的平均加速比分别为 1.41,1.16,1.22.Features ANN 获得了最好的预测性能.因此,Features ANN 优化顺序预测模型在 2 种平台上都获得了平均最佳的预测性能.Features ANN 能够获得较好程序性能的原因在于模型基于机器学习算法构建,能在较短的时间内生成新程序的最佳优化顺序.而使用现有迭代编译方法为新程序搜索最佳优化顺序时,虽然现有方法在优化搜索性能和搜索时间上具有较大的改进,但仍需要进行多次迭代才能获得局部最佳程序性能.

5.3　Features ANN 与非迭代编译方法的比较

非迭代编译方法的典型代表是基于机器学习的编译优化方法和基于多面体模型的编译优化方法,本文选择与 Ashouri 等人[24]和 Park 等人[25]的研究进行对比分析.Ashouri 等人[24]提出 COBAYN 优化顺序预测模型,COBAYN 是基于贝叶斯网络构建的优化顺序预测模型.Park 等人[25]基于多面体编译框架,采用静态程序特征和几种不同的机器学习算法构建优化顺序预测模型.采用 Features ANN 和现有非迭代编译方法预测优化顺序时,预测模型性能如图 10 和图 11 所示.其中,COBAYN 表示 Ashouri 等人[24]提出的优化顺序预测模型,LR 和 SVM 表示 Park 等人[25]基于多面体框架通过逻辑回归和支持向量机构建的预测模型,Features ANN 表示本文提出的优化顺序预测模型.

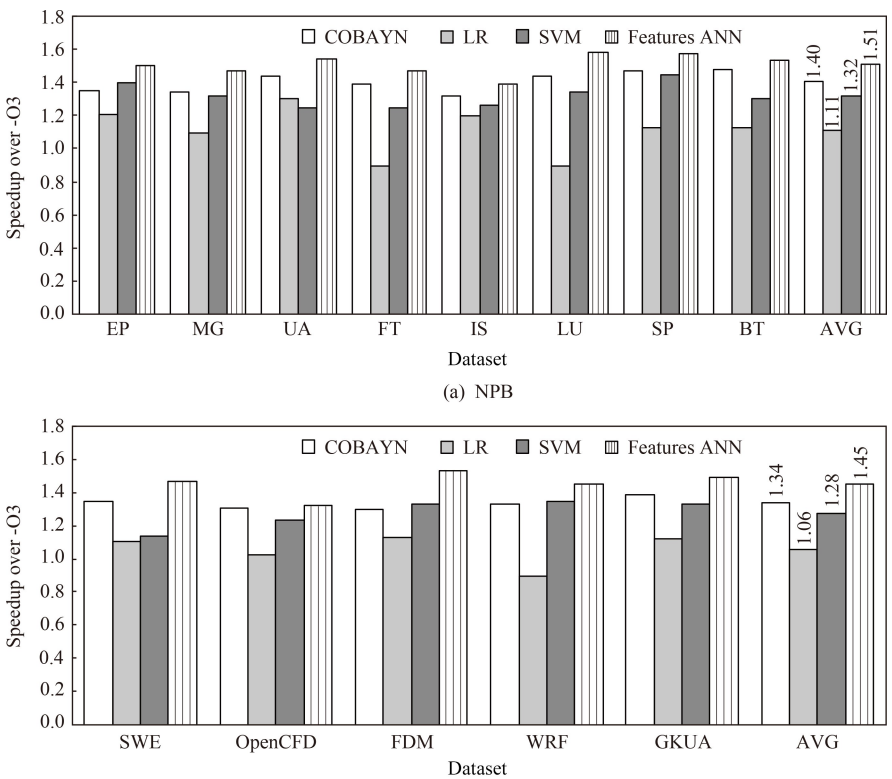
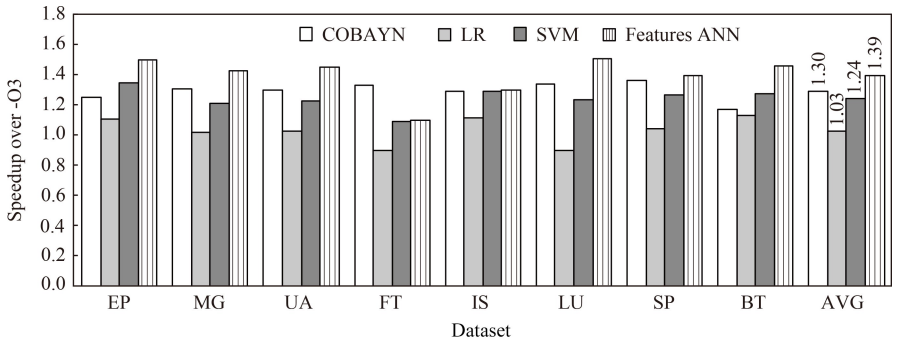
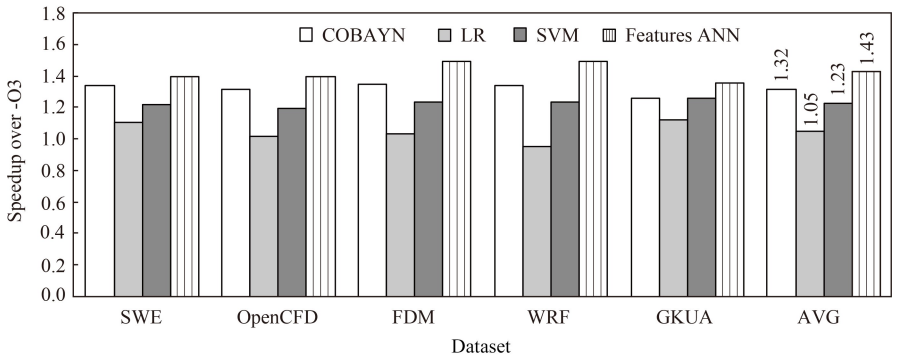


Fig. 10　Comparison with non-iterative compilation (platform 1)

图 10　Features ANN 与非迭代编译方法的性能比较(平台 1)



(a) NPB



(b) Scientific computing program

Fig.11 Comparison with non-iterative compilation (platform 2)

图 11 Features ANN 与非迭代编译方法的性能比较(平台 2)

从图 10 可以看出,平台 1 上基于非迭代编译方法为 NPB 测试集和大型科学计算程序预测优化顺序时,Features ANN 相对于 GCC 标准优化级别-O3 的平均加速比分别为 1.51 和 1.45,COBAYN 的平均加速比分别为 1.40 和 1.34,LR 的平均加速比分别为 1.11 和 1.06,SVM 的平均加速比分别为 1.32 和 1.28.从图 11 可以看出,平台 2 上基于非迭代编译方法为 NPB 测试集和大型科学计算程序预测优化顺序时,Features ANN 相对于 GCC 标准优化级别-O3 的平均加速比分别为 1.39 和 1.43,COBAYN 的平均加速比分别为 1.30 和 1.32,LR 的平均加速比分别为 1.03 和 1.05,SVM 的平均加速比分别为 1.24 和 1.23.对当前测试用例而言,Features ANN 具有最好的预测性能,COBAYN 次之,SVM 的预测性能低于 COBAYN,LR 具有相对较低的预测性能.Features ANN 能够获得较好预测性能的原因在于模型采用 PCA 特征降维技术,选择最主要的程序特征作为预测模型的输入.COBAYN 采用探索性因子分析法选择程序特征,然而在实际应用中探索性因子分析必须和验证性因子分析结合使用,才能更好地进行因子分析.LR 和 SVM 均采用固定的程序特征,缺乏程序特定性.此外,Features ANN 基于 ANN 根据当前状态下的程序特征选择最佳优化,而现有方法对整个程序优化前的特征进行抽取和预测优化,Features ANN 的预测精度更高.

5.4 离线学习和在线预测时间开销比较

使用 Features ANN 预测模型为新程序预测优化顺序时的细粒度时间,分为训练阶段(离线完成)和预测阶段(在线完成).训练阶段一次性完成,收集训练数据所需时间,取决于训练集中应用程序数量.表 4 和表 5 分别表示平台 1 和平台 2 上 SPEC CPU 作为训练集、WRF 作为目标应用程序时不同预测模型每个特定阶段所需时间.采用本文提出的方法虽然需要若干天时间进行数据收集和模型训练,但对于新的目标程序需要很短的时间就可以完成较好的优化参数预测.

实验结果表明,Features ANN 预测模型的离线学习时间略高于现有方法,但在线预测时间低于现有方法.而且从 5.1 节和 5.3 节的分析中可以看到,Features ANN 在特征选择、预测性能等方面均优于现有方法.因此,Features ANN 在特征选择、预测性能和在线预测时间开销等方面均取得了较好的效果.

Table 4　Learning and Prediction Time of WRF (Platform 1)

表 4　WRF 离线学习和在线预测时间(平台 1)　s

Method	Data Collection	Model Construction	Feature Extraction	Model Prediction	Average Time Cost
GraphDSE	9.50×10^5	676		235	9.50×10^5
ClusterDSE	10.37×10^5	548		489	10.38×10^5
LR	8.64×10^5	234	11.5	0.79	8.64×10^5
SVM	9.50×10^5	456	12.1	0.86	9.50×10^5
COBAYN	10.37×10^5	475	10.7	0.98	10.37×10^5
Features ANN	10.37×10^5	393	11.1	0.45	10.37×10^5

Table 5　Learning and Prediction Time of WRF (Platform 2)

表 5　WRF 离线学习和在线预测时间(平台 2)　s

Method	Data Collection	Model Construction	Feature Extraction	Model Prediction	Average Time Cost
GraphDSE	9.50×10^5	686		246	9.51×10^5
ClusterDSE	10.37×10^5	573		491	10.38×10^5
LR	8.64×10^5	249	11.7	0.78	8.64×10^5
SVM	9.50×10^5	478	13.1	0.83	9.50×10^5
COBAYN	10.37×10^5	493	15.7	0.96	10.38×10^5
Features ANN	10.37×10^5	399	14.1	0.43	10.37×10^5

6　结束语

本文提出一种选择编译器优化顺序的新方法:Features ANN,以最大化目标应用程序性能.该方法通过使用动静结合的特征表示方法进行程序表示,在最小均方误差意义下获取对原始特征数据的表示.基于程序特征和当前状态最佳优化构成优化顺序预测模型的样本数据,采用人工神经网络建立统计模型并集成于编译器框架,实施优化顺序选择及进行编译过程驱动.在 2 种平台上采用动静结合特征作为 Features ANN 预测模型输入时,相对于 GCC 5.4 编译器默认标准优化级别-O3 分别获得 1.49x 和 1.41x 的程序执行时间加速.与现有迭代编译方法 GraphDSE 和 ClusterDSE,及非迭代编译方法 COBAYN,LR,SVM 比较时,均获得了最佳的性能提升.

程序优化顺序选择是提升程序性能的关键技术,目前仍有一些后续工作值得研究,具体来说包括:1)我们将考虑增加更多的训练数据集、测试数据集和目标平台,以进一步提升预测准确率和通用性.2)由于不同的程序优化可能需要不同的目标代码特征,我们将研究更多优化变换的相关性特征,进一步提升程序性能.3)尝试人工添加噪声的方法增加预测方法的鲁棒性,研究在负载严重的多用户环境下提升预测性能.

参　考　文　献

[1] Ashouri H A, Bignoli A, Palermo G, et al. MiCOMP: Mitigating the compiler phase-ordering problem using optimization sub-sequences and machine learning [J]. ACM Transactions on Architecture and Code Optimization, 2017, 14(3): No.12

[2] Nickolls J, Dally W J. The GPU computing era [J]. IEEE Micro, 2010, 30(2): 56-69

[3] Duran A, Klemm M. The Intel© many integrated core architecture [C] //Proc of the 2012 Int Conf on High Performance Computing and Simulation. Piscataway, NJ: IEEE, 2012: 365-366

[4] The National Supercomputing Guangzhou Center. The TOP500 list of the world's supercomputers [OL]. 2017 [2017-06-19]. http://www.chinastor.com/hpc-top500/201706/0Q43H562017.html

[5] Rajam A S, Clauss P. The polyhedral model of nonlinear loops [J]. ACM Transactions on Architecture and Code Optimization, 2015, 12(4): No.48

[6] Shirako J, Hayashi A, Sarkar V. Optimized two-level parallelization for GPU accelerators using the polyhedral model [C] //Proc of the 26th Int Conf on Compiler Construction. New York: ACM, 2017: 22-33

[7] Lu Pingjing. Research on key techniques in low-cost iterative compilation optimization [D]. Changsha: National University of Defense Technology, 2010 (in Chinese)
(陆平静. 低开销的迭代编译优化关键技术研究[D]. 长沙:国防科技大学, 2010)

[8] Chen Yang, Fang Shuangde, Eeckhout L, et al. Practical iterative optimization for the data center [J]. ACM SIGARCH Computer Architecture News, 2012, 40(1): 49-60

[9] Purini S, Jain L. Finding good optimization sequences covering program space [J]. ACM Transactions on Architecture and Code Optimization, 2013, 9(4): No.56

[10] Nobre R, Martins L G A, Cardoso J M P. A graph-based iterative compiler pass selection and phase ordering approach [C] //Proc of the 17th ACM SIGPLAN/SIGBED Conf on Languages, Compilers, Tools, and Theory for Embedded Systems. New York: ACM, 2016: 21-30

[11] Martins L G A, Nobre R, Cardoso J M P, et al. Clustering-based selection for the exploration of compiler optimization sequences [J]. ACM Transactions on Architecture and Code Optimization, 2016, 13(1): No.8

[12] Huang Qijing, Lian Ruolong, Canis A, et al. The effect of compiler optimizations on high-level synthesis for FPGAs [C] //Proc of the 21st Int Symp on Field Programmable Custom Computing Machines. Piscataway, NJ: IEEE, 2013: 89-96

[13] Ballal P A, Sarojadevi H, Harsha P S. Compiler optimization: A genetic algorithm approach [J]. International Journal of Computer Applications, 2015, 112(10): 9-13

[14] Liu Hui, Zhao Rongcai, Wang Qi. Function level compiler optimization parameters selection method based on supervised learning model [J]. Computer Engineering and Science, 2018, 40(6): 957-968 (in Chinese)
(刘慧, 赵荣彩, 王琦. 监督学习模型指导的函数级编译优化参数选择方法研究[J]. 计算机工程与科学, 2018, 40(6): 957-968)

[15] Ashouri A H. Compiler autotuning using machine learning techniques [D]. Toronto, Canada: University of Toronto, 2016

[16] Park E, Kulkarni S, Cavazos J. An evaluation of different modeling techniques for iterative compilation [C] //Proc of the 14th Int Conf on Compilers, Architectures and Synthesis for Embedded Systems. Piscataway, NJ: IEEE, 2011: 65-74

[17] Agakov F, Bonilla E, Cavazos J, et al. Using machine learning to focus iterative optimization [C] //Proc of the 4th Int Symp on Code Generation and Optimization. Piscataway, NJ: IEEE, 2006: 295-305

[18] Fursin G, Kashnikov Y, Wahid A, et al. Milepost GCC: Machine learning enabled self-tuning compiler [J]. International Journal of Parallel Programming, 2011, 39(3): 296-327

[19] Cavazos J, Fursin G, Agakov F, et al. Rapidly selecting good compiler optimizations using performance counters [C] // Proc of the 5th Int Symp on Code Generation and Optimization. Piscataway, NJ: IEEE, 2007: 185-197

[20] Dubach C, Jones T M, Bonilla E V, et al. Portable compiler optimization across embedded programs and microarchitectures using machine learning [C] //Proc of the 42nd Int Symp on Microarchitecture. New York: ACM, 2009: 78-88

[21] Hoste K, Eeckhout L. Cole: Compiler optimization level exploration [C] //Proc of the 6th Int Symp on Code Generation and Optimization. Piscataway, NJ: IEEE, 2008: 165-174

[22] Kumar T S. Optimizing code by selecting compiler flags using parallel genetic algorithm on Multicore CPUs [J]. International Journal of Engineering and Technology, 2014, 6(2): 544-551

[23] Jantz M R, Kulkarni P A. Exploiting phase inter-dependencies for faster iterative compiler optimization phase order searches [C] //Proc of the 2013 Int Conf on Compilers, Architectures and Synthesis for Embedded Systems. Piscataway, NJ: IEEE, 2013: No.7

[24] Ashouri A H, Mariani G, Palermo G, et al. COBAYN: Compiler autotuning framework using Bayesian networks [J]. ACM Transactions on Architecture and Code Optimization, 2016, 13(2): No.21

[25] Park E, Cavazos J, Pouchet L N, et al. Predictive modeling in a polyhedral optimization space [J]. International Journal of Parallel Programming, 2013, 41(5): 704-750

[26] Wang Guochang, Cheng Guojian, Carra T. The application of improved neuroevolution of augmenting topologies neural network in marcellus shale lithofacies prediction [J]. Computers & Geosciences, 2013, 54: 50-65

Liu Hui, born in 1982. PhD, lecturer. Her main research interests include high performance computing, compiler optimization, machine learning.

Xu Jinlong, born in 1985. PhD, lecturer. His main research interests include high performance computing, compiler optimization, machine learning.

Zhao Rongcai, born in 1957. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include high performance computing, compiler optimization, program performance optimization.

Yao Jinyang, born in 1992. Master. His main research interests include high performance computing, compiler optimization.