文章编号：2095-2163(2023)11-0286-04 中图分类号：TP314 文献标志码：A

基于强化学习的循环自动展开研究

李居垚，何先波
（西华师范大学 电子信息工程学院，四川 南充 637009）

摘 要：近年来，芯片行业蓬勃发展，新的架构比以往更快地涌现出来。同时，现代计算机的应用场景变得越来越复杂，对计算机性能的要求逐渐增加。在编译优化中，循环展开发挥着承上启下的作用，且任务复杂，高度依赖经验，并需要大量人力和资源投入。为了减少编译器开发中循环展开的工作量，并适应芯片行业快速发展的环境，本文提出了一种基于强化学习的自动展开器。经过实验比较，该循环展开器性能优于 Clang -O3，并且与蛮力搜索相比具有更快的编译速度。
关键词：循环展开；编译器；自动生成；强化学习

Research on loop automatic unfolding based on reinforcement learning

LI Juyao, HE Xianbo
(College of Electronics and Information Engineering, China West Normal University, Nanchong Sichuan 637009, China)

Abstract: In recent years, the chip industry has flourished, and new architectures have emerged at the faster pace than before. At the same time, the application scenarios of modern computers are becoming increasingly complex, and the requirements for computer performance are gradually increasing. Loop unrolling in compilation optimization plays a connecting role, and it is a complex task that highly relies on expert experience and requires a significant investment of manpower and resources. To reduce the workload of loop unrolling in compiler development and adapt to the rapidly developing environment of the chip industry, this article proposes an automatic unroller based on reinforcement learning. After experimental comparison, the proposed unroller performs better than Clang -O3, and it also has a faster compilation speed compared to brute force search.
Key words: loop unrolling; compiler; automatically; reinforcement

0 引 言

当下新架构芯片问世的频率远超以往，这需要缩短芯片编译的开发周期来进行适配。在编译器中许多关键决策是由抽象模型来决定的，虽然这些模型能够有效地解决问题，但是开发编写一个精确的抽象模型需要耗费大量的人力物力；所以使用新技术来缩短编译器开发的成本和周期是有必要的。

编译优化中的循环展开有着承上启下的作用，因为循环展开从多维度间接影响了系统性能，同时也会为其他优化（如 SLP）带来契机，所以开发一种有别于传统的新技术来对编译器循环展开进行自动化是必要的。本文将使用人工智能技术提出一个模型来对编译器循环展开工作中的至关重要的展开因子（Unroll Factor，UF）进行自动生成，已达到循环展开工作自动化减少编译器开发工作量和成本的目的。

1 相关理论及方法

1.1 循环展开

循环展开是一种常用的编译优化策略，开启后编译器会通过既定的抽象模型启发式地将目标循环进行多次复制。循环展开主要用于为其他优化提供机会，例如，展开会创建与动态执行相对应的多个静态内存单一操作指令，可以重新安排这些指令以利用内存局部性[1]。如果循环访问连续迭代中的相同内存位置，引用还可以用标量进行消除。循环展开是暴露相邻内存引用的关键，展开完成后后续的优化可识别到这些引用并将其合并为一个宽引用[2]。

由于循环展开的主要影响集中在次级效应上，所以选取最优的展开因子进行循环展开是困难的。循环展开也可以被归纳为空间换时间的一种手段，这很容易造成一种循环展开总能提升性能的误解。不合适的循环展开都会造成以下性能损失：会降低

作者简介：李居垚（1995-），男，硕士研究生，主要研究方向：编译器中端优化。
通讯作者：何先波（1971-），男，博士，教授，主要研究方向：嵌入式系统。Email：983510960@qq.com
收稿日期：2023-04-01

指令缓存的性能，会造成内存的溢出和重新加载，如果编译器选择推测性的访问展开内存会造成动态冲突[3]。循环展开会增加编译时间，所以通过暴力搜索等手段获取循环展开因子是低效的，图 1 展示了不同循环展开因子选择对编译耗时的影响。

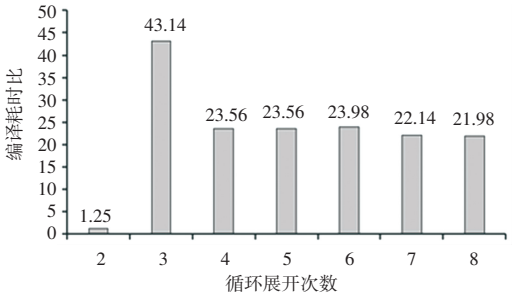

图 1　TSVC 中 s3111.c 设置不同循环展开次数的编译耗时比

Fig. 1　Compilation time ratio of s3111. c with different loop unwinding times in TSVC

使用暴力搜索的方式来寻找最优的展开因子是效率低下的，如图 1 所示。图 1 中，以不进行展开的耗时为基准。近年来，随着机器学习技术不断发展成熟，许多学者希望利用机器学习来解决复杂的编译器优化问题，即构建一个机器学习模型去预测编译器优化参数和编译选项等[4]。本文将会尝试使用强化学习来预测的循环展开因子。

1.2　强化学习

强化学习通过和环境动态交互不断试错来得到较优解的方法。在这种方法中智能体通过不断和环境交互来进行学习[5]，强化学习想要直接训练得到策略函数或者价值函数是比较困难的，对此引入神经网络来近似目标函数是一种常用的策略。

在强化学习中智能体通过对环境的“观察”来采取行为动作，行为会产生新的环境并根据当前环境得到一个奖励的分值作为行为好坏的评判，整个学习过程的目的是奖励达到数学期望的最大化并得到相映的决策函数（π^*）：

$$\pi^* = arg_\pi \max \mathbb{E}_{\tau\sim\pi(\tau)}[\tau] \tag{1}$$

强化学习有很多巧妙的算法，本文选用 PPO（Proximal Policy Optimization）算法[6]。PPO 算法的优势在于每次进行梯度更新时能够使得成本最小化，并且能够很好地消除和上一次梯度更新的偏差。

1.3　代码嵌入

使用代码词嵌入能够使输入的源程序更好地通过强化学习进行训练，代码词嵌入的最终目标是有效地将源程序和强化学习所需要的向量进行映射。在本文中，选择 IR2Vec 作为代码嵌入。

IR2Vec[4] 是一种基于源码中间表示形式（LLVM IR）的代码嵌入，这种代码嵌入的方法具有简明和可伸缩的特点，IR2Vec 通过表征学习和流信息相结合的方式捕获程序的语法和语义将程序表示为连续空间中的分布式嵌入。IR 的表示被转化为一个种子嵌入词汇表，并且有 2 种编码模式：符号编码和流感知编码。由于本文主要针对的是源代码中循环部分的工作，控制流的信息至关重要，所以本文使用 IR2Vec 的流感知编码模式。

2　相关工作

2.1　实验平台

实验平台信息见表 1。

表 1　实验平台信息

Tab. 1　Experimental platform information

配置内容	配置参数
使用语言	Python
操作系统	Linux
CPU	Intel(R) Xeon(R) CPU E5-2630 v4@2.20 GHz
内存	503Gi
平台架构	X86
编译器版本	LLVM11

2.2　实验内容

本次实验所使用的模型如图 2 所示，源程序作为初始的输入会通过脚本识别代码中的循环并插入循环展开指令设置循环展开因子，然后使用 Clang\LLVM 将源代码转为中间表示形式（IR），而后将 IR 输入到 IR2Vec 中生产长度为 300 的向量嵌入到强化学习的智能体中，同时编译运行使用 LLVM 基准（LLVM-O3）的可执行文件并收集运行时间。下一阶段使用策略函数输出的新因子作为参数，再次进行编译并运行，2 次运行的性能差异将作为模型的奖励，这意味着本模型的奖励不需要手动进行设置，奖励的值也不是固定不变的。模型中一轮迭代后会将预测出的循环展开因子作为下一轮的学习所需要的编译选项参数，获得新的 IR、新的向量嵌入智能体，以此往复训练出预测循环展开因子的模型。

本次实验将会采取上述的模型和方法进行，代码嵌入使用的 IR2Vec 开源工具并根据设定的任务要求进行改造，研究中选用其流感知模式来更好地识别循环中的控制流信息。在搭建强化学习模型的过程中，使用了 RLlib[6] 和 Tune[7]，RLlib 和 Tune 都能在强化学习的开源库里找到。这样一来，研究所搭建的模型会更加规范，便于进行参数的调整和对模型的扩展。

在实验中使用 Ray[8] 框架，因此对于提前设置

好的 3 种不同的学习率可以同时进行,这极大地缩短了训练所需的时间。同时,为了与本次研究搭建的强化学习模型进行对比,还准备了暴力搜索来锁定最优循环展开因子。

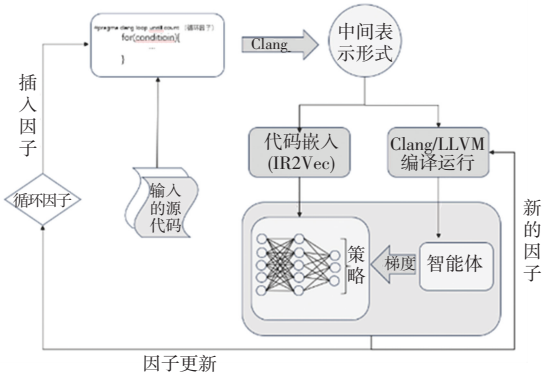

图 2 本文提出的自动循环展开框架

Fig. 2 The automatic loop unrolling framework proposed in this article

2.3 参数设置

实验选用 PPO 算法,设置了 3 个学习率:5-e3、5-e4、5-e5。本模型的奖励将以 LLVM11 -O3 编译后的运行时间作为基准,将其进行如下定义:

$$reward = RunTime_{llvm-o3} - RunTime_{RL} / RunTime_{llvm-o3}$$

对于循环展开因子的取值,本文设立一个数组 array[1、2、4、…、8],此后随机从中取值作为强化学习的动作。

表 2 是本文构建的强化学习模型具体的参数设置。

表 2 强化学习模型参数

Tab. 2 Parameters of reinforcement learning model

模型参数	数值
sample_batch_size	25
train_batch_size	500
sgd_minibatch_size	20
num_sgd_iter	20
learning - rate	5-e3,5-e4,5-e5

2.4 数据集

实验所使用的训练集是在向量化测试套件(TSVC)的基础上进行筛选和拓展组成的训练集。TSVC 最初由 Callahan 等人开发后由 Maleki 等学者[9]将其从 Fortran 转换为 C 进行了扩展。扩展版本有 151 个内核。本次研究选择这个基准测试是因为其中提供了一些相对简单的循环的集合,这些循环可以在许多科学用途的 C 语言代码中找到,同时 TSVC 中的循环能够根据其测试的用途进行适当分类,对结果分析有很大的帮助。通过脚本将 TSVC 中每个函数提取出去生成独立的 C++文件;TSVC 的数据量对强化学习来说还是偏小了,本实验将会在 TSVC 原有的循环上进行增加,手段主要是改变变量名称使得数据的总量倍增;同时,对不同分类的 TSVC 函数进行随机抽取用来构建测试集。构建训练集和其中一个测试集如图 3 所示。

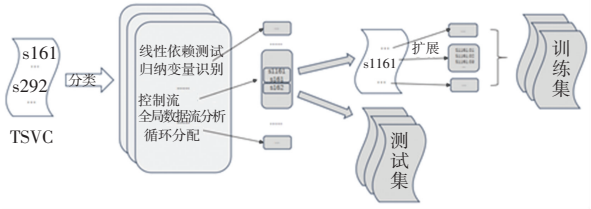

图 3 TSVC 训练集和测试集构建流程

Fig. 3 TSVC training and testing set construction process

为了测试本文构建的强化学习循环展开化器能否很好地推广到新代码,并且测试其对于循环复杂程度的敏感程度,选择使用 Ameer 等学者[10]构建的循环结构简单的数据集 Tests、分离出的 TSVC 内核和 PloyBench[11]三个测试集进行推广性能的测试。这 3 个测试集中,Tests 的循环结构最简单,其次是 TSVC,PloyBench 的循环结构最为复杂其主要由循环构成,即由多个 benchmark 组成。本次研究选用其中的 2mm、bicg、atax、gemm、gemver、choleshy 来进行测试。

3 实验结果

强化学习循环展开器、LLVM 基准模型和暴力搜索的性能对比如图 4 所示。在 Tests 基准上本文提出的循环展开器比 LLVM 基准模型有 1.19x 的性能提升;在 TSVC 基准上,有 1.13x 的性能提升;在 Polybench 基准上,有 1.09x 的性能提升。可以看到,随着循环复杂度的提升,循环展开器的性能也随之下降,可见在面对复杂循环时本文提出的循环展开器还有进一步的优化空间。

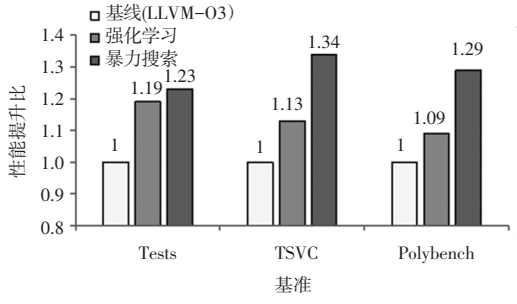

图 4 归一化后强化学习循环展开器、LLVM 基准模型和暴力搜索的性能比

Fig. 4 The performance ratio of reinforcement learning loop unroller, LLVM benchmark model and brute force search after normalization

虽然本文提出的循环展开器性能上要弱于暴力搜索的结果，但是在运行速度上大幅领先暴力搜索，如图 5 所示：在 Tests 基准上，本文提出的循环展开器比暴力搜索快 5.97x；在 TSVC 基准上，快 6.23x 的性能提升；在 Polybench 基准上快 5.61x 的性能提升。

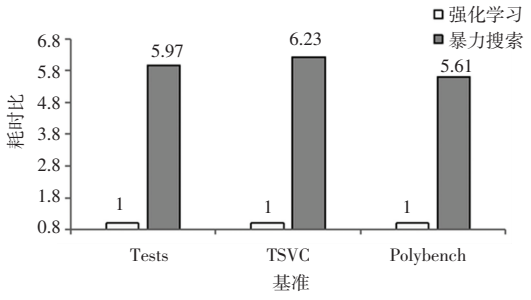

图 5　归一化后强化学习向量器和暴力搜索编译所需时间比

Fig. 5　Time ratio between reinforcement learning vector machine and brute force search compilation after normalization

4　结束语

针对编译开发成本急需缩减的问题，本文提出了一个基于强化学习的循环展开器来对编译开发中循环展开进行自动化。通过代码嵌入的方式来表征循环代码，然后使用强化学习模型来对展开因子进行预测以达到输入新代码能快速自动地进行循环展开的目的。实验表明，本文提出的循环展开器相比于 LLVM -O3 的基准模型有更好的效果，同时有比暴力搜索更快的编译速度。但根据本文选取的 3 个测试集的实验结果，本文提出的向量化器在复杂场景中会效果下降，在未来的工作中会考虑更多复杂应用场景，将谓词加入到控制流的识别当中、将多面体模型和强化学习进行结合以提升循环展开器的性能。

参考文献

[1] DAVIDSON J W, JINTURKAR S. Memory access coalescing: A technique for eliminating redundant memory accesses [C]// Proceedings of the SIGPLAN '94 Conference on Programming Language Design and Implementation. Orlando, FL: ACM, 1994: 186-195.

[2] LARSEN S, AMARASINGHE S. Exploiting superword level parallelism with multimedia instruction sets[J]. ACM SIGPLAN Notices, 2000, 35(5): 145-156.

8[3] STEPHENSON M, AMARASINGHE S. Predicting unroll factors using supervised classification [C]//Proceedings of the International Symposium on Code Generation and Optimization (CGO '05). Vancouver, DC: IEEE Computer Society, 2005: 123-134.

[4] 池昊宇，陈长波. 基于机器学习的编译器自动调优综述[J]. 计算机科学，2022，49(1)：241-251.

[5] JOHN S, WOLSKI F, DHARIWAL P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv: 1707.06347, 2017.

[6] LIANG E, LIAW R, NISHIHARA R, et al. Ray rllib: A composable and scalable reinforcement learning library[J]. arXiv preprint arXiv:1712.09381, 2017.

[7] LIAW R, LIANG E, NISHIHARA R, et al. Tune: A Research Platform for Distributed Model Selection and Training[J]. arXiv preprint arXiv:1807.05118, 2018.

[8] MORITZ P, NISHIHARA R, WANG S, et al. Ray: A distributed framework for emerging AI applications[C]//Proceedings of the 13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18). Carlsbad, CA, USA: USENIX, 2018: 561-577.

[9] CALLAHAN D, DONGARRA J, LEVINE D. Vectorizing compilers: A test suite and results [C]//Supercomputing, Supercomputing' 88. Orlando, FL, USA: IEEE Computer Society Press, 1988: 98-105.

[10] AMEER H A, AHMED N K, WILLKE T, et al. Neuro Vectorizer: end-to-end vectorization with deep reinforcement learning[C]// Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization (CGO 2020). New York, NY, USA: ACM, 2020: 242-255.

[11] POUCHET L N. Polybench: The polyhedral benchmark suite [EB/OL]. [2012]. http://www.cs.ucla.edu/pouchet/software/polybench.