计算机科学
COMPUTER SCIENCE

一种基于TVM的自动调度搜索优化方法

韩林, 王一帆, 李嘉楠, 高伟

引用本文

韩林, 王一帆, 李嘉楠, 高伟. 一种基于TVM的自动调度搜索优化方法[J]. 计算机科学, 2025, 52(3): 268-276.

HAN Lin, WANG Yifan, LI Jianan, GAO Wei. Automatic Scheduling Search Optimization Method Based on TVM[J]. Computer Science, 2025, 52(3): 268-276.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240100126

一种基于 TVM 的自动调度搜索优化方法

韩 林[1] 王一帆[2] 李嘉楠[1] 高 伟[1]
1 郑州大学国家超级计算郑州中心 郑州 450001
2 郑州大学计算机与人工智能学院 郑州 450001
(hanlin@zzu.edu.cn)

摘 要 随着人工智能的迅猛发展,新型算子与硬件不断涌现,算子库的开发和维护面临着巨大的挑战,仅仅依靠手工优化已无法满足 AI 模型性能提升的需求。Ansor 是一种基于 TVM 的算子自动调度技术,可以针对不同的后端搜索深度学习模型或算子的最佳调度方案,生成高性能代码而无需用户手动定义模板,但其巨大的搜索空间造成了搜索效率低下的问题。因此,提出了两种优化方案:1)基于强化学习的算法实现最佳性能草图的选择;2)基于机器学习模型的突变规则预测。两种优化方案旨在缩短最佳调度方案的搜索时间,快速生成高性能的算子。为评估优化方案的有效性,对 Resnet-50 等 3 种模型和 conv2d 等 3 种算子进行测试与评估。结果显示,优化后的 Ansor 只用 70%~75%的搜索时间就可以生成性能与之前相同甚至更优的目标程序,并且在最佳迭代次数下,目标程序的推理速度最高可提升 5%。

关键词:自动调度;TVM 编译器;搜索速度优化;机器学习;强化学习;深度学习模型
中图分类号 TP311

Automatic Scheduling Search Optimization Method Based on TVM

HAN Lin[1],WANG Yifan[2],LI Jianan[1] and GAO Wei[1]
1 National Supercomputing Center in Zhengzhou,Zhengzhou University,Zhengzhou 450001,China
2 School of Computer and Artificial Intelligence,Zhengzhou University,Zhengzhou 450001,China

Abstract With the rapid development of artificial intelligence and the continuous emergence of new operators and hardware,the development and maintenance of operator libraries face enormous challenges. Relying solely on manual optimization can no longer meet the needs of improving AI model performance. Ansor is an operator automatic scheduling technique based on TVM,which can search for the best scheduling schemes for different backend deep learning models or operators,generate high-performance code without the need for users to manually define templates. However,the huge search space results in low search efficiency. Therefore,two optimization schemes are proposed. One is to select the optimal performance sketch based on Reinforcement learning algorithm,and the other is to predict mutation rules based on machine learning models. Two optimization schemes aim to reduce the search time for the optimal scheduling scheme and quickly generate high-performance operators. To evaluate the effectiveness of the optimization plan,three models such as Resnet-50 and three operators such as conv2d are tested and evaluated. The results show that the optimized Ansor can generate target programs with the same or even better performance as before in only 70%~75% search time. Moreover,under the optimal iteration number,the inference speed of the target program can be improved by up to 5%.

Keywords Auto schedule,TVM complier,Optimizing search speed,Machine learning,Reinforcement learning,Deep learning model

1 引言

深度学习是人工智能领域中的一种重要技术,已被广泛应用于图像识别、自然语言处理、语音识别等领域。深度学习算法通常需要大量的计算资源来进行训练和推理,因此如何提高计算效率成为了深度学习领域的一个热门问题。大部分厂商的做法是定制高性能算子库,如 CUDNN[1],MIOpen[2]。这些算子库为深度学习算子的高效实现提供支持,使得深度学习模型可以在 GPU 等计算硬件上提高运行速度。虽然高性能算子库可以大大提高深度学习算法的计算效率,但是实现这些算子库往往需要深度学习领域的专家进行定制,针对具体的硬件和算法进行优化,这无疑将消耗巨大的人力和物力。并且由于缺乏专业知识和技术经验,开发者很难设计和实现高效的深度学习算法。因此,高性能算子库的使用范围

到稿日期:2024-01-12 返修日期:2024-06-23
基金项目:河南省重大科技专项(221100210600)
This work was supported by the Major Science and Technology Special Project of Henan Province(221100210600).
通信作者:高伟(yongwu22@126.com)

受到了一定的限制[3]。

深度学习编译器的出现在一定程度上解决了手工开发高性能算子库的问题。深度学习编译器可以自动将模型转换为高效的代码,大大提高了深度学习算法的计算效率[4],使得深度学习算法可以在更广泛的硬件平台上运行,而不仅仅局限于专门的高性能计算硬件。目前,出现了 TVM[5],AKG[6],MLIR[7]等深度学习编译器。以 TVM 为例,其支持多种前端语言,如 Tensorflow[8],Pytorch[9],ONNX[10]等,可以统一将这些深度学习模型转换成高级中间表示 Relay IR[11],通过算子融合、常量折叠等优化遍对 IR 进行优化,优化后的 IR 可以降级为张量表达式。张量表达式基于 Halide[12]思想,使计算定义与调度分离。AutoTVM[13]和 Ansor[14]可以自动搜索最佳调度策略,并自动生成高性能代码,从而部署在各个硬件平台上[15]。深度学习编译器可以自动进行优化和调整,用户无需花费大量时间和精力来进行算法的优化和调整,能够更加专注于算法的设计和应用,从而提高效率。Ansor 作为基于 TVM 的自动调度框架,通过若干推导规则自动生成代码的高层结构,并在此基础上通过随机注释生成完整的程序。其中,程序集形成搜索空间,并通过遗传算法对搜索空间进行择优筛选,得到最适用于当前场景的调度方案。对于较大的模型而言,Ansor 需要对其进行任务拆分,通过找到各个子任务的最佳调度方案来确定整体的最佳调度策略。对模型的调度实际上是对若干个子任务的调度,Ansor 需要对每一个子任务进行搜索空间的生成,并通过多轮遗传算法的迭代,才能找到最佳的调度策略。但是,由于搜索空间庞大且搜索空间中的程序是随机突变生成的,因此程序性能并不稳定,甚至大部分程序的性能较差。因此,Ansor 需要较长的搜索时间以及足够多的迭代次数,才能找到最佳的调度策略。

针对 Ansor 搜索最佳调度方案的时间过长的问题,本文提出了改进的 Explore-Then-Commit(ETC)算法用于草图预测,以及基于机器学习模型的规则预测。前者使用了改进的强化学习算法,通过少量的尝试,综合已得到的程序性能,找到性能表现更优的高层结构,并在此基础上生成整体性能更优的程序集;后者将训练一个机器学习代价模型用于突变规则的预测,使得程序朝着收益最大的方向突变,加快程序搜索的收敛。这两种优化方法将明显缩短 Ansor 的搜索时间,并且可以略微提升生成的调度策略的性能。

本文的主要贡献如下:

1)提出了一种基于改进 ETC 算法的草图预测方法;

2)提出了一种基于代价模型的规则预测方法;

3)基于 TVM 开源深度学习编译器,设计并实现了以上两种优化方法;

4)在若干深度学习算子以及深度学习模型上进行了测试与分析,验证了上述方法的有效性。

2　背景介绍

2.1　自动调度技术

Ansor 是一种基于 TVM 的自动调度技术,通过在搜索空间中找到最优解来生成最佳调度方案。本文提出了一种基于推导的枚举方法,通过递归地应用几个基本推导规则来生成若干个草图,这些草图是最适合当前程序的高级结构。但是草图不包含并行化、分块大小等细节,是不完整的程序。然后,Ansor 对草图执行随机注释,进行细节的填充,生成成千上万个用于微调和评估的完整程序,由此形成一个庞大的程序集。

Ansor 使用基于机器学习的代价模型对搜索空间中的程序的运行时间进行评估,并使用进化搜索配合代价模型来择优筛选出一批性能最好的候选集。将候选集作为测试集在真实机中进行测量,将测量的真实运行时间作为标签值对代价模型进行更新,同时记录性能表现最优的程序。

接下来将重新基于草图进行随机注释,生成搜索空间,并将上一轮的候选集也加入搜索空间,以提高搜索空间程序的整体性能。程序使用代价模型在更优的搜索空间进行进化搜索,再一次生成新的候选集。由于代价模型的准确率更高且搜索空间程序性能更好,理论上后一轮生成的候选集会比前一轮候选集的性能更好,因此将得到更优的调度方案。按以上步骤循环迭代,直到程序运行结束,最终生成最优的调度方案。Ansor 的整体流程如图 1 所示。

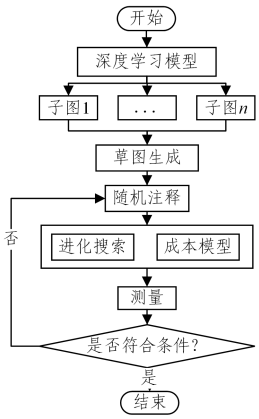

图 1　Ansor 的整体流程

Fig. 1　Overall process of Ansor

2.2　强化学习

强化学习是一种机器学习方法,用于描述和解决智能体在与环境的交互过程中通过学习策略来达成回报最大化或实现特定目标的问题。在强化学习中,智能体在不同的状态下做出不同的行动,从而获得奖励或惩罚。强化学习是智能体以试错的方式进行学习,通过与环境进行交互获得的奖赏来指导行为,其目标是使智能体获得最大的奖赏,进而实现收益最大化。

强化学习的基本框架包括状态、行动、奖励和策略。状态表示环境的状态,行动表示智能体可以采取的行动,奖励表示智能体在某一状态下采取某一行动所获得的奖励或惩罚,而策略表示智能体在不同状态下选择行动的规则。智能体通过对环境的判断形成价值函数,使用策略对价值函数进行判断,使得智能体在整个过程中获得最大的奖励,如图 2 所示。

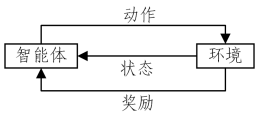

图 2　强化学习框架

Fig. 2　Reinforcement learning framework

2.3 XGBoost 模型

XGBoost[16]是一种梯度提升树模型。它是一种高效的、可扩展的机器学习算法，被广泛应用于各种数据科学竞赛和工业应用中。该模型使用梯度提升算法来训练决策树模型，通过对训练集进行迭代，每次迭代都训练一个新的决策树模型来提高模型性能。在每次迭代中，该模型会根据之前决策树模型的预测结果来对训练集进行加权，使得模型更加关注之前预测错误的数据点，这样模型就可以逐步学习到更加准确的预测规则。XGBoost 模型的优点在于其具有很好的泛化能力和鲁棒性，可以处理大规模数据集和高维特征，并且能够自动处理缺失值、异常值和噪声数据。此外，它还支持并行计算，可以利用 GPU 等硬件进行加速，因此在训练大规模模型时非常高效。

XGBoost 的核心算法思想是：不断地添加树，不断地进行特征分裂来生长一棵树，每次添加一棵树其实是学习一个新函数 $f(x)$，去拟合上次预测的残差。当训练完成得到 k 棵树后，预测一个样本的分数，实际上就是根据这个样本的特征，计算出对应的叶子节点，每个叶子节点就对应一个分数。每棵树对应的分数之和就是该样本的预测值。

3 基于机器学习的突变规则预测

3.1 遗传算法分析

Ansor 在完成随机注释后，会生成一个庞大的搜索空间，然后代价模型会提取搜索空间中程序的特征，如分块大小、访存次数等，通过这些特征预测该程序的运行时间。利用代价模型预测程序运行时间的方法可以快速地评估搜索空间中所有程序的运行时间，降低了程序在真实机中运行测量的时间成本。

然后，程序会选取搜索空间中的部分优秀程序作为候选集，记录结果并将其加入到下一轮的搜索空间中。通过不断地迭代，搜索空间中程序的质量会逐渐提高，最终可以得到接近全局最优解的程序。在迭代过程中，Ansor 选用了遗传算法，通过对原始种群进行一些特征的交叉和变异，再通过代价模型的评估，不仅可以得到更加优异的种群，而且可以扩大程序集，使程序集表现出多样性，便于找到全局最优解。遗传算法的流程如图 3 所示。

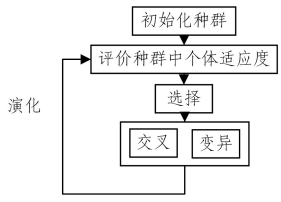

图 3 遗传算法的流程

Fig. 3 Genetic algorithm process

在 Ansor 的遗传算法中，有若干条突变规则，包括分块大小的突变、是否并行化、是否向量化等。现有的程序在突变完成之后，程序的特征也会随之变化，代价模型将会重新对程序进行评估，观察其性能的变化。Ansor 为每条突变规则都设置了一个权重，代表程序触发当前突变规则的可能性。对于每个程序而言，触发突变规则是完全随机的，这意味着程序将会根据初始权重随机触发这些突变规则并完成突变。对于某一个程序而言，它可能触发性能收益高的突变规则，也有可能触发性能负收益的突变规则，虽然增加了随机性，可以更好地寻求全局最优解，但是不利于搜索过程的快速收敛。对于某些程序而言，可能会导致搜索很长时间才能得到性能提升较大的程序。若每个程序在突变时都可以选择最合适的突变方向完成突变，则程序集的整体性能质量就可以得到提高。因此，训练一个机器学习模型，使得在遗传算法中，针对每一个程序，都会根据程序的特征选择其收益最高的突变方向，指引该程序朝着这个突变方向完成突变，突变生成新的程序。放弃让程序随机突变，因为这样可能会导致负优化。而且张量程序也不会陷入局部最优解，因为代价模型会根据张量程序的特征来判断它更适合于哪种突变。如果张量程序的某个循环轴迭代次数过多，而当前的分块大小设置是不合理的，则重新设置分块的大小总是最有利的，代价模型会指引其朝着分块的突变方向进行突变；如果张量程序已经拥有合适的分块大小，则应该转向其他方向进行突变，代价模型会根据张量程序的具体特征来判断应该如何突变，这种突变方向总是最适合于当前张量程序的。因此加快了收敛速度，使得通过更少的迭代次数就可以搜索到性能更优的程序。

本节提出了基于代价模型的规则预测，使用预训练模型，根据程序的特征确定最佳的突变方向，使得程序朝着收益最高的方向完成突变，优化搜索空间程序性能，实现缩短搜索时间的目标。

3.2 构建机器学习模型

首先，对机器学习模型完成预训练，其关键在于应准备好程序的特征和与之对应的标签，再完成模型的构建。该实验的特征和标签分别是程序特征和该程序的收益最高的突变方向所对应的索引。准备好这两项数据后，就可以完成多分类机器学习模型的构建。文中选取常用的模型以及常用的算子作为预训练的数据集，如 Resnet 系模型、CNN 模型等，以及常见的算子组合，如矩阵乘加、卷积算子等。Ansor 基于算子融合规则，把模型分为多个组(Group)，这些组在递降的过程中会变成若干子图，Ansor 以子图为单位进行自动调度。这些子图中的算子都是按照算子融合的规则进行划分的，一般多个 element 算子符合融合规则，可被分到一个子图中；而 opaque 算子一般不能与其他算子融合，算子本身就是一个子图，子图中的算子类型会呈现一定的相似性。因此，训练好的机器学习模型在预测新子图的突变方向时，由于子图算子类型的相似性，也能保持较高的预测准确率。将这些准备好的模型和算子使用 Ansor 进行自动调度，使其朝着所有的方向进行突变并记录突变结果，为特征的提取和标签值的选取做准备。模型构建的整体流程如图 4 所示。

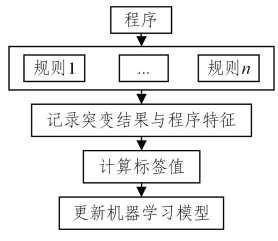

图 4 模型构建

Fig. 4 Model construction

3.2.1　特征提取

对于程序特征的提取,需要尽可能地覆盖程序的所有特点,从而使预测结果更加精确。为了确保特征提取的全面性和有效性,选择 Ansor 本身针对程序提取的特征作为本文算法的程序特征,并在此特征值上决定标签值的选取。在遗传算法中,调用评估函数时,会生成当前搜索空间的所有特征,因此可以得到搜索空间程序的特征作为机器学习模型构建的特征。这样做不仅可以实现程序特点的全覆盖,而且使用相同的特征有利于后续处理。

3.2.2　标签值确定

确定了程序的特征后,需要确定特征对应的标签值。在本文算法中,标签值代表当前特征程序的突变方向,为了程序可以朝着收益最高的方向突变,标签值应该是收益最高的突变方向所对应的索引。

为了拿到有效的标签值,选择适当的若干个算子及模型,使用默认 Ansor 为其生成高性能程序。在这个过程中需要对默认 Ansor 进行一些功能的修改,使得随机注释过程中搜索空间中的各个程序都向所有方向完成突变,并记录该程序朝着各个方向突变后的分值,这个分值将作为标签值选择的重要依据。

对于该分值,综合突变规则权重等信息,通过合适的决策手段,选出最适合于当前程序突变方向的索引,并将其作为当前程序的标签值。完成数据的收集后,就可以使用该数据集对代价模型完成训练,并完成模型的构建工作。当收集到特征和标签后,只需要预设一些超参数,即可开始进行代价模型的训练工作,训练时间本身的开销并不会影响实验本身,实验仅需要代价模型进行推理工作即可。

选择 xgboost 模型,并且通过上述准备好的数据集完成模型的训练,进而获得一个训练好的 xgboost 模型。可以通过模型根据程序的特征预测其收益最高的突变方向,以优化搜索空间。

3.3　突变规则预测

当机器学习模型构建完成后,通过输入一个程序的特征,就能得到其收益最高的突变方向。突变规则的过程主要在进化搜索阶段进行,旨在选取搜索空间中每个程序的最佳突变方向。当搜索空间的每个程序都朝着收益最高的方向突变,搜索空间程序的整体性能将会提升,并且随着迭代的进行,将会形成正反馈,有利于搜索过程的快速收敛,使得程序可以用更短的时间完成整个搜索过程。

机器学习模型对程序进行规则预测的示意图如图 5 所示。

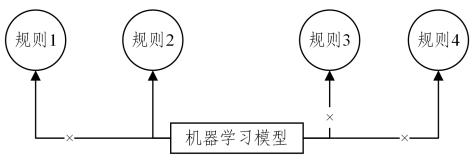

图 5　规则预测示意

Fig. 5　Schematic of rule prediction

从图 5 可以看到,当机器学习模型完成对某个程序的预测后,可以预测出规则 2 是当前程序收益最高的突变方向。因此舍弃其他突变方案,记录当前程序的最佳突变方向,为后续的突变收集信息,并在后续的突变过程中使用该索引作为突变方向。

规则预测的算法流程如算法 1 所示。

算法 1　规则预测

输入:features / * 当前搜索空间中的程序集 * /

输出:states / * 原程序集突变后的程序集 * /

1. xgboost=GetXgboost() / * 获得训练好的机器学习模型 xgboost * /
2. labels=xgboost(features) / * 预测每个程序收益最高的突变方向 * /
3. for i in range(0,population) do
4.　　label=labels[i]/ * 当前程序的突变方向 * /
5.　　rule.Apply(states[i]) / * 对当前程序进行突变操作 * /
6. end for
7. return states / * 突变后的程序集 * /

当程序进入随机突变的步骤时,应先收集搜索空间中每个程序的最佳突变方向,在获悉了每个程序收益最高的突变方向后遍历每个程序,控制该程序朝着最佳突变方向进行突变,就可以最大程度地提高程序的性能。

4　基于改进 ETC 算法的草图预测

4.1　改进 ETC 算法

ETC 算法是强化学习中的一种探索-利用策略。在强化学习中,探索指采取新的行动以获取更多的环境信息,而利用指根据已有的知识和经验做出最佳决策。

ETC 算法的基本思想是在开始阶段进行探索,收集有关环境和行动的信息,然后根据所获得的信息做出决策。具体来说,ETC 算法通常会在一段时间内进行探索,期间会尝试各种可能的行动,以便了解环境的奖励结构和行动的效果。然后根据收集到的信息,选择最佳的行动策略进行利用,以获得最大的累积奖励。

ETC 算法的关键是在探索和利用之间找到一个平衡。在探索阶段,它会牺牲一些即时的奖励来获取更多的信息,以便在后续的利用阶段做出更好的决策。ETC 算法的整体步骤如图 6 所示。

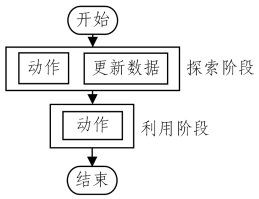

图 6　ETC 算法

Fig. 6　ETC algorithm

ETC 算法的探索和利用阶段是相互独立的,在经历一段时间的探索后,便会一直使用之前探索的经验来进行利用动作。这样做的好处是,当事件经过了充分的探索后,可以按照探索给出的正确经验进行利用动作。但是如果探索轮数不够充分,则会导致探索带来的经验并不准确,并且可能会误导利用的动作,直到最后结束;而如果探索的轮数足够多,利用的轮数则会减少,累计收益也会减少。探索越多,利用带来的累计收益就越少。因此,本文根据具体的使用场景对 ETC 算法进行了改进并将其应用到当前场景。

对于任意算子或者子任务,Ansor 都会根据规则推导出若干草图,然后通过随机注释随机选择草图并填充调度细节

以及分块大小，以生成具体的程序。但是基于不同草图生成的程序性能表现不一，对于某个算子而言，生成的其中一个草图可能很适合于当前的硬件结构，但是生成的另一个草图并不适合于当前的硬件结构。因此，如果随机选择草图，可能会不断地基于某些性能表现不佳的草图生成程序，作为搜索空间程序集的一部分，而由于搜索空间的大小是有限的，因此这样会减少基于优秀草图生成的程序的数量，从而减少发现更优调度方案的可能性。在 Ansor 的执行过程中可以发现，对于算子而言，草图的生成只会执行一次，生成之后便缓存起来，这就意味着整个过程中都是在相同的草图结构上补充细节。对于已生成的多个草图，希望可以一直选择最优的高层结构，并在此基础上生成搜索空间，这样程序就有更多机会在最优草图上进行进化搜索，理论上会生成更加优秀的程序集，使得程序调度的效果更好。

首先，针对以上应用场景，本文对 ETC 算法进行了改进。为了确保有足够的利用收益，应该减少探索的轮次，同时避免因为探索不充分而可能得到错误的经验。其次，我们不希望探索得到的经验会一直影响利用的动作，因为可能存在一些偶然场景使得探索得到了错误的经验，这样将会一直误导利用的动作，反而降低生成算子的性能。因此，对利用阶段进行了改进，在利用阶段也会同时进行探索，这样做的好处是，经过初步探索后，一般程序都能得到正确的动作，也会继续选择正确的动作进行利用。如果初步探索得到了错误的经验，也可以通过后续利用阶段的探索做进一步的纠正。通过这样的方法，可以用更少的探索次数得到更多的利用收益。改进的 ETC 算法的整体流程如图 7 所示。

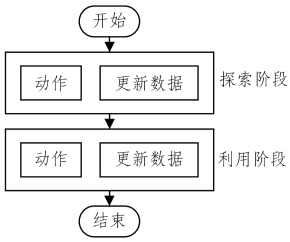

图 7 改进的 ETC 算法

Fig. 7 Improved ETC algorithm

通过在随机选择草图前调用决策函数，来确定前一轮迭代中将要选择的草图，并基于该草图生成搜索空间，从而达到优化搜索空间的目的，最终实现优化搜索时间的目标。并且应用了改进的 ETC 算法，使得程序使用最少的探索轮次得到了最高的利用收益，由于在利用阶段也会进行探索，因此避免了某些偶然情况导致的性能损失。

4.2 代价模型预热

草图预测的思路在于，在 Ansor 的迭代过程中对中间结果进行收集，并综合自定义策略完成下一轮迭代的草图选择，但是需要考虑 Ansor 本身的一些特点进行调整。

由于在 Ansor 运行的过程中，对代价模型采用的是边训练边更新的策略，因此在最开始的几轮迭代过程中，代价模型并不能准确地预测程序的执行时间。随着迭代轮数的增加，对代价模型进行多次更新后，才能有一个相对较高的预测准确率。而在代价模型的性能达到一定水平之前，进化搜索是没有意义的，因为进化搜索筛选最优程序的依据是代价模型，如果代价模型无法准确地预测程序的运行时间，则筛选出的就不是最优程序。随着代价模型的准确率达到一定水平后，进化搜索才能搜索到最优程序，此时生成的测量数据才有利用的价值。因此，要先进行代价模型的预热，使其具有较好的性能。

实验显示，程序在第三轮搜索时开启草图预测表现最佳，此时代价模型的准确率已经足够高，而且程序会有更多的轮次调整草图选择方案。草图预测的整体流程如图 8 所示。

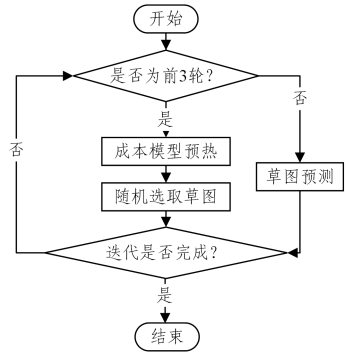

图 8 草图预测的整体流程

Fig. 8 Overall process of sketch prediction

为了确保吞吐量的准确性，需要对成本模型先进行预热处理，待成本模型的性能稳定后有较高的准确率时，程序才开始收集吞吐量信息，这样才能更准确地评估草图的性能。

在成本模型预热完成后，开始草图预测。为了增加程序的容错率，降低偶然情况导致的性能下跌的风险，本优化方法将对每一轮次的结果进行收集，并且历史数据也会在每一轮次的搜索过程中进行更新，再针对更新后的历史数据做决策，以更好地指引草图预测，提高程序预测的稳定性与正确性。

4.3 草图预测

在成本模型预热完成后，程序进入探索阶段。探索阶段将在第四轮搜索开启，程序将会随机选取草图，并基于这些草图做随机注释，通过成本模型评估其性能，进行数据的收集。经过一轮的探索后，程序进入利用阶段，程序将会根据探索阶段收集的先验信息选择当前性能最优的草图。经测试发现，平均吞吐量是最能反映某个草图性能的指标，因此程序选用平均吞吐量来衡量草图性能的优劣。并且在利用阶段，程序也需要收集吞吐量等信息来对历史信息进行更新，这样就符合改进 ETC 算法的思想，避免探索不充分导致的先验信息不正确。草图预测步骤如图 9 所示。

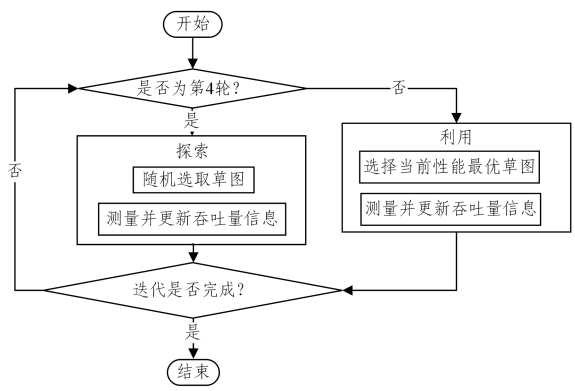

图 9 草图预测步骤

Fig. 9 Sketch prediction

程序在进行成本模型预热后，进入草图预测的第一个轮次，即第四轮是没有吞吐量信息的，因此需要先进行一次随机选取草图，获取这一轮次的吞吐量信息。之后的每一个轮次都可以对上一轮的吞吐量信息以及历史吞吐量信息进行加权计算，得到当前性能最优的草图，这样就可以只基于性能最优的草图做随机注释，并更新历史平均吞吐量。

草图预测算法如算法 2 所示。

算法 2　草图预测

```
输入：throughtputs / * 每个草图的平均吞吐量 * /
输出：states / * 基于目标草图生成的搜索空间的程序集 * /
1. means=GetThroughput() / * 获得每个草图的平均吞吐量 * /
2. index=DoPolicy(means) / * 根据决策函数得到草图序号 * /
3. for i in range(0,population) do
4.    state=sketches[index] / * 选择目标草图建立高层结构 * /
5.    states.push(state)6. end for
7. / * 对 state 进行随机注释并返回 * /
8. Update(means) / * 根据当前程序集的吞吐量更新历史平均吞吐量 * /
```

5　实验评估

5.1　研究问题

本文提出了两种优化方法，用于解决 Ansor 搜索时间较长的问题。通过这两种优化方法，旨在对比在吞吐量一定的情况下，与优化前的 Ansor 相比，优化后的程序是否可以有效地缩短搜索时间。对于缩短搜索时间的测试，将通过加速比来体现与衡量。

5.2　实验平台

本文使用编译器 TVM1.1 对程序进行测试，CPU 的型号为 Intel core i7-7700HQ，系统为 ubuntu18.04，GPU 的型号为 NVIDIA GTX1060，显存为 6 GB，内存为 8 GB。运行环境如表 1 所列。

表 1　运行环境

Table 1　Runtime environment

CPU	Intel core i7-7700HQ
操作系统	ubuntu 18.04
GPU	NVIDIA GTX1060 6 GB
内存	8GB DDR4

实验基于 CPU 平台进行，采用 llvm12 作为后端编译器。实际上两种优化方法的主要目的是加快最佳调度策略的搜索速度，并不会显式地优化调度策略本身，因此在 CPU 平台下，选择官方推荐的后端编译器 llvm12 进行实验。

5.3　数据集

由于 Ansor 对模型和算子的处理方法不同，为了确保优化方案的普适性，实验将在若干个模型和算子上进行测试，并与优化前的 Ansor 进行对比，以确保优化方案在模型和算子上都有效果。实验选择在深度学习领域中高频出现的模型和算子作为测试数据集，确保算法的优化是有意义的。本次实验选择了 Resnet-50，Resnet-18，Mobilenet V2 这 3 个模型，并选取了卷积算子和批量矩阵乘法算子。算子尺寸的大小随机指定，取结果的平均值作为测试结果。

5.4　测量结果

本次实验的目的在于验证优化后的 Ansor 程序能否用更短的搜索时间达到与优化前的 Ansor 几乎相同的推理性能。为了测试模型与算子的性能，本实验对常见的模型和算子都分别进行了测试，并记录每个模型和算子的运行时间。记 $T\sim$ 为模型或算子在默认的 Ansor 中生成的调度方案的吞吐量达到最佳调度方案的搜索时间，记 T 为模型或算子在优化后的 Ansor 中达到与前者调度方案的吞吐量几乎相同情况下的搜索时间。为了方便对比两者的差异，使用时间比值 S 的形式。假定在默认 Ansor 中所有模型和算子的时间比值恒定为 1，通过计算优化后的 Ansor 的模型和算子的时间比值，来对比两者的搜索时间的差异。时间占比的计算式为：

$$S=\frac{T}{T\sim} \tag{1}$$

图 10 分别给出了二维卷积运算、一维卷积运算和批矩阵相乘 3 种算子的实验结果。横坐标代表算子的种类，纵坐标代表算子搜索时间与默认 Ansor 搜索时间的时间比值。其中蓝色的柱子代表在默认 Ansor 上的执行结果，时间比值恒定为 1；橙色的柱子代表在优化后的 Ansor 上的执行结果，其数值代表与默认 Ansor 的搜索时间比值，计算方法见式(1)。可以看出，3 类算子在吞吐量相同的情况下，优化后的搜索速度均优于优化前的搜索速度，其中二维卷积算子的加速效果最为明显，仅用 74% 的时间就达到了默认 Ansor 搜索到的最佳方案的吞吐量。3 类算子的平均时间占比为 75%。可以看出，对于常见的算子而言，本文提出的两种优化策略是有效的。

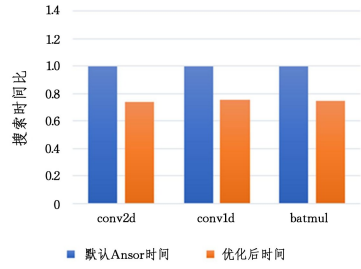

图 10　算子搜索时间占比(电子版为彩图)

Fig. 10　Proportion of operator search time

图 11 分别给出了 Resnet-50，Resnet-18，Mobilenet V2 这 3 个模型的实验结果。横坐标代表模型的种类，纵坐标代表模型搜索时间与默认搜索时间的比值。其中蓝色的柱子代表在默认 Ansor 上的执行结果，时间比值恒定为 1；橙色的柱子代表在优化后的 Ansor 上的执行结果，其数值代表与默认 Ansor 的搜索时间比值，计算方法见式(1)。

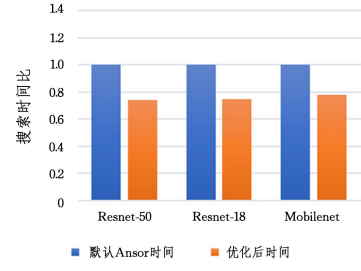

图 11　模型搜索时间占比(电子版为彩图)

Fig. 11　Proportion of model search time

对于 3 类模型而言，在确保吞吐量相同的情况下，优化后的搜索速度均优于优化前的搜索速度，其中 Resnet-50 模型

的速度提升最大，仅用 74%的搜索时间就达到了默认 Ansor 搜索到的最佳调度方案的吞吐量。3 种模型的平均搜索时间比值为 77%。可以看出，对于常见模型而言，本文提出的两种优化策略依然是有效的。

为了验证优化后的 Ansor 程序对推理速度也有一定提升，分别对 Conv1d，Conv2d，batch_mul 算子以及 Resnet-50，Resnet-18，Mobilenet V2 模型进行了测试。测试的目的在于，验证优化后的 Ansor 程序是否可以在同样的调优轮次下达到更快的推理速度。为了确保实验的正确性和合理性，本实验的调优轮次都设置为官方推荐的调优轮次。对于 3 种算子而言，观察其在 1000 调优轮次下的推理速度表现；对于 3 种模型而言，分别采用 12000，15000，20000 的调优轮次，并对比其在最佳调优轮次下的推理速度表现。

同样，为了突出实现的对比效果，本次实验采用时间比值的方式对推理速度进行量化，默认 Ansor 对于算子以及模型的推理速度比值恒定为 1，优化后的 Ansor 对于算子以及模型的推理速度数值为与默认推理速度数值的比值。图 12 给出了在最佳调优轮次下，优化后的程序的算子在推理速度上与默认 Ansor 程序的对比。

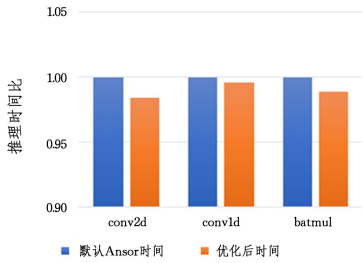

图 12 算子推理速度对比

Fig. 12 Comparison of operator inference speeds

图 13 给出了在最佳调度轮次下，优化后的程序的模型在推理速度上与默认 Ansor 程序的对比。

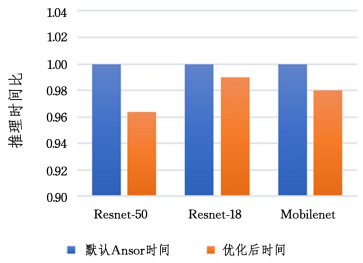

图 13 模型推理速度对比

Fig. 13 Comparison of model inference speed

优化后的 Ansor 在推理速度方面未能带来巨大收益，因为在官方推荐的调优轮次下，搜索出的调度方案已经接近最优解。但是也有提升效果，主要体现在优化后的 Ansor 使用更少的轮次就能达到更好的调优效果。图 14 给出了 Conv2d 在默认 Ansor 与优化后的 Ansor 中在相同迭代轮次下的优化效果对比。

图 15 给出了 Resnet-50 模型在默认 Ansor 与优化后的 Ansor 中在相同迭代轮次的优化效果对比。

图 16 给出了 Resnet-18 模型在默认 Ansor 与优化后的 Ansor 中在相同迭代轮次的优化效果对比。

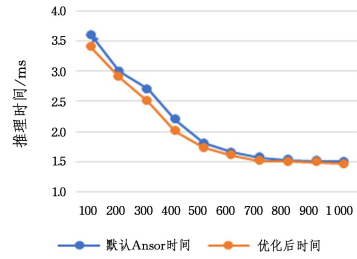

图 14 Conv2d 在不同调优轮次的性能

Fig. 14 Performance of Conv2d with different tuning rounds

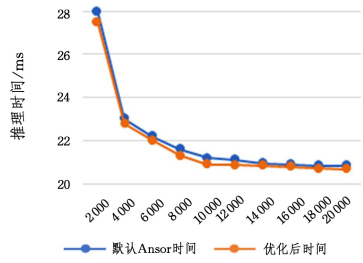

图 15 Resnet-50 在不同调优轮次的性能

Fig. 15 Performance of Resnet-50 with different tuning rounds

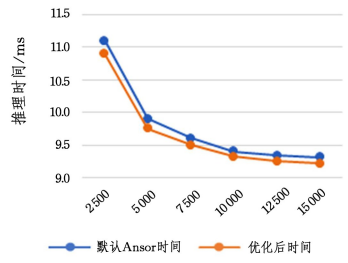

图 16 Resnet-18 在不同调优轮次的性能

Fig. 16 Performance of Resnet-18 with different tuning rounds

由图 14—图 16 可以看出，在不同轮次下，优化后的 Ansor 的调度方案均比默认 Ansor 的调度方案的性能更好，并且优化后的 Ansor 用更少的调优轮次就能达到默认 Ansor 使用官方推荐调优轮次的效果。

为了论证两种优化方案的收益，分别只使用突变规则预测和草图预测作用于以上 3 种模型，观察各自优化的搜索时间占比，确定两种优化方案各自的收益。

图 17 给出了只采用突变规则预测的 Ansor 在对 Resnet-50，Resnet-18，Mobilenet V2 这 3 个模型进行自动调度时的搜索时间占比，分别为 0.88，0.87 和 0.92，平均仅用了 89%的搜索时间就可以搜索到默认 Ansor 生成的最佳调度方案。

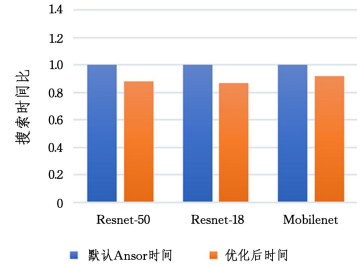

图 17 仅用突变规则预测的搜索时间占比

Fig. 17 Proportion of search time predicted solely by mutation rules

图 18 给出了只采用草图预测的 Ansor，在对 Resnet-50，

Resnet-18，Mobilenet V2 这 3 个模型进行自动调度时的搜索时间占比，分别为 0.79，0.82 和 0.83，平均仅用 82% 的搜索时间就可以搜索到默认 Ansor 生成的最佳调度方案。

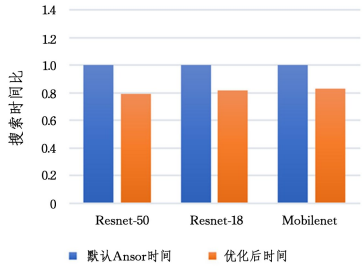

图 18　仅用草图预测的搜索时间占比

Fig. 18　Proportion of search time predicted solely by sketches

而综合了两种优化方案后的 Ansor 在搜索时间占比上的表现更好，从理论上也符合预期。

5.5　实验分析

本节将针对本次实验，对 3 种模型和 3 种算子的性能提升进行详细的分析。首先简述模型与算子在自动调度中的区别。相较于算子而言，模型在 Ansor 中的步骤会多一步，即将模型划分为不同子任务，然后每次将针对性能可能提升最大的子任务进行调度。子任务的划分实际上是在算子融合这一步骤确定的，算子融合相关 Pass 会根据融合规则将符合特定规则的算子划分到一个组中。因此在算子融合后，整个模型实际上被划分成了多个组，而这些组在递降的过程中会一直保留，这里的子任务实际上就是组在递降到 Ansor 阶段的另一种等价形式。子任务实际上是由单个或多个算子构成的，而对一个模型进行调度，本质上是对多个子任务分别进行调度，一个模型的搜索时间实际上是多个子任务的搜索时间总和。此外，需要说明的是，子任务的划分是由算子融合相关 Pass 确定的，而在 Ansor 对子任务的自动调度阶段，更多的是对子任务中的算子做一些内存或指令方面的优化。而本文提出的两种优化方法主要是提升后者搜索最佳调度策略的收敛速度，因此对于由不同的融合规则形成的子图都是有效的。

因此，在本次实验中，模型搜索时间的缩短本质上是因为其中的多个子任务的搜索时间的缩短。接下来，仅探究两种优化方法是如何使子任务的搜索时间缩短的。

两种优化方法的目的在于提升搜索空间中程序集的性能，使得在搜索次数较少的情况下可以搜索到吞吐量较高的调度方法。在默认的 Ansor 程序中，通过描述计算的定义进行自动调度，然后进行草图生成、随机注释、遗传算法、测量与模型更新等步骤，其中需要重点关注随机注释以及遗传算法步骤。

在随机注释中，其基于已有的若干个草图，并在此基础上进行细节填充。这些高层结构存在性能表现的差异，如果基于某个草图上生成的程序性能普遍不佳，则应当尽可能地规避这个高层结构，更多地基于其他的草图生成程序，这样就可以生成更好的程序集。换言之，也可以根据多次初始测量结果，确定基于某个草图生成的程序性能总是最好的，这样程序就可以只基于当前的草图生成程序，组成总体性能表现最好的程序集。为了保证程序的健壮性，避免偶然导致的在某个性能不佳的草图上生成了性能不错的程序，误导初始测量结果，我们还添加了每一轮的探索，避免了一错到底的情况，如果发现错误，则会及时调整。

而在遗传算法中，搜索空间的程序将会按照 Ansor 给定的权重进行随机突变，这些突变可能会使得程序吞吐量增加，也可能会导致程序吞吐量减少，但是如果程序可以朝着收益最高的方向突变，程序就能更快地收敛。因此，将根据程序的特征，通过判断它朝哪个方向突变后的收益最高来敲定突变方向，使得每个程序都朝着收益最高的方向突变。在收益几乎相同的情况下，将利用权重等信息，通过算法来确定其最优突变方向。

上述两种优化策略分别在随机注释阶段和突变阶段提高了搜索空间中程序集的性能。由于搜索空间的大小是一定的，如果搜索空间中程序集的整体性能提高，就可以更快地寻求到算子的最优调度方案，从而花费较少的搜索时间就能得到更好的调度效果。

结束语　本文提出了改进的 ETC 算法和基于机器模型的规则预测两种方法，用于优化 Ansor 在搜索最佳调度方案时搜索时间过长的问题。使用优化后的 Ansor 程序，针对深度学习中出现的高频算子和模型进行了测量，并进行实验结果的收集，主要通过观察优化后的加速比来分析优化提升的大小。实验结果表明，在大部分算子和模型中，在确保吞吐量几乎不变的情况下，与默认 Anosr 相比，两种优化方法都可以大幅缩短搜索时间。

参 考 文 献

[1] CHETLUR S, WOOLLEY C, VANDERMERSCH P, et al. cudnn: Efficient Primitives for Deep Learning[J]. arXiv: 1410.0759, 2014.

[2] KHAN J, FULTZ P, TAMAZOV A, et al. MIOpen: An Open Source Library for Deep Learning Primitives[J]. arXiv: 1910.00078, 2020.

[3] LI M Z, LIU Y, LIU X Y, et al. The Deep Learning Compiler: A Comprehensive Survey[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 32(3): 708-727.

[4] XING Y, WENG J, WANG Y S, et al. An In-depth Comparison of Compilers for Deep Neural Networks on Hardware[C]// 2019 IEEE International Conference on Embedded Software and Systems (ICESS). IEEE, 2019: 1-8.

[5] CHEN T Q, MOREAU T, JIANG Z H, et al. TVM: End-to-End Optimization Stack for Deep Learning[C]// Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI'18). Carlsbad, USA: USENIX Association, 2018: 579-594.

[6] ZHAO J, LI B J, WANG N, et al. AKG: Automatic Kernel Generation for Neural Processing Units Using Polyhedral Transformations[C]// Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2021). New York, USA: Association for Computing Machinery, 2021: 1233-1248.

[7] LATTNER C, AMINI M, BONDHUGULA U, et al. MLIR: Scaling Compiler Infrastructure for Domain Specific Computa-

tion[C]// 2021 IEEE/ACM International Symposium on Code Generation and Optimization(CGO). IEEE, 2021: 2-14.

[8] ABADI M, BARHAM P, CHEN J M, et al. Tensorflow: Large-scale Machine Learning on Heterogeneous Distributed Systems [C]// Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation(OSDI'16). USA: USENIX Association, 2016: 265-283.

[9] PASZKE A, GROSS S, MASSA F, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library[C]// Proceedings of the 33rd International Conference on Neural Information Processing Systems. Red Hook, NY, USA: Curran Associates Inc., 2019: 8026-8037.

[10] GASKILL B. ONNX: the Open Neural Network Exchange Format[J]. Linux Journal, 2018, TN. 285: 157-161.

[11] ROESCH J, LYUBOMIRSKY S, WEBER L, et al. Relay: A New IR for Machine Learning Frameworks[C]// Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL 2018). New York, NY, USA: Association for Computing Machinery, 2018: 58-68.

[12] RAGAN-KELLEY J, BARNES C, ADAMS A, et al. Halide: A Language and Compiler For Optimizing Parallelism, Locality, And Recomputation in Image Processing Pipelines[J]. ACM Sigplan Notices, 2013, 48(6): 519-530.

[13] CHEN T Q, ZHENG L M, YAN E, et al. Learning to Optimize Tensor Programs[C]// Proceedings of the 32nd International Conference on Neural Information Processing Systems(NIPS'18). USA: Curran Associates Inc., 2018: 3393-3404.

[14] ZHENG L M, JIA C F, SUN M M, et al. Ansor: Generating High-Performance Tensor Programs for Deep Learning[C]// Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI'20). Carlsbad, CA, USA: USENIX Association, 2020: 863-789.

[15] WU J J. A deployment method and device for heterogeneous platforms based on TVM compiler: CN202010654954[P]. 2023-12-25.

[16] CHEN T Q, GUESTRIN C. XGBoost: A Scalable Tree Boosting System[C]// Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). New York, NY, USA: Association for Computing Machinery, 2016: 785-794.

HAN Lin, born in 1978, Ph.D, associate professor, is a senior member of CCF (No. 16416M). His main research interests include compiler optimization and high-performance computing.

GAO Wei, born in 1988, Ph.D. His main research interests include AI complier optimation and advanced compilation technology.

(责任编辑:柯颖)