计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(12):112−119 [doi: 10.15888/j.cnki.csa.008880]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于多任务深度学习的 HXDSP 多簇软流水研究①

刘纯纲, 周　鹏, 郑启龙

(中国科学技术大学 计算机科学与技术学院, 合肥 230026)
通信作者: 郑启龙, E-mail: qlzheng@ustc.edu.cn

摘　要: 针对目前编译优化领域的深度学习模型普遍采用单任务学习而难以利用多个任务间的相关性提升模型整体编译加速效果的问题, 提出了一种基于多任务深度学习的编译优化方法. 该方法使用图神经网络 (GNN) 从 C 程序的抽象语法树 (ASTs) 和数据控制流图 (CDFGs) 中学习得到程序特征, 然后对程序特征同步预测 HXDSP 软件流水启动间隔和循环展开因子. 在 DSPStone 数据集上的实验结果表明, 该多任务方法取得了相对于单任务方法 12% 的性能提升.

关键词: 软件流水; 循环展开; 多任务学习; 图神经网络; 编译优化

引用格式: 刘纯纲,周鹏,郑启龙.基于多任务深度学习的 HXDSP 多簇软流水研究.计算机系统应用,2022,31(12):112–119. http://www.c-s-a.org.cn/1003-3254/8880.html

Research on Multi-cluster Software Pipelining of HXDSP Based on Multi-task Deep Learning

LIU Chun-Gang, ZHOU Peng, ZHENG Qi-Long

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China)

Abstract: The current deep learning models in the field of compilation optimization generally perform single-task learning and fail to use the correlation among multiple tasks to improve their overall compilation acceleration effect. For this reason, a compilation optimization method based on multi-task deep learning is proposed. This method uses the graph neural network (GNN) to learn program features from the abstract syntax trees (ASTs) and control data flow graphs (CDFGs) of the C program and then predicts the initiation interval and loop unrolling factor for the software pipelining of the HX digital signal processor (HXDSP) synchronously according to program features. Experimental results on the DSPStone dataset show that the proposed multi-task method achieves a performance improvement of 12% compared with that of the single-task method.

Key words: software pipelining; loop unrolling; multi-task learning; graph neural network (GNN); compilation optimization

1　引言

HXDSP[1] 是中国电子科技集团第 38 所研制的一款国产分簇结构的超长指令字 (VLIW) 数字信号处理器 (DSP), 有着强大的并行计算和数据读写能力. HXDSP 处理器拥有 4 个执行簇和 4 条数据总线, 每个簇上有 8 个算术逻辑单元器 (ALU), 8 个乘法器 (MUL), 4 个移位器 (SHF), 1 个超算器 (SPU) 以及一组寄存器. DSP 应用程序数据并行度高, 通常包含着很多循环, 对循环的优化可以极大提高应用程序的执行效率. 循环的优化技术一般有向量化、软件流水和循环展开等[2]. 目前的编译器通常运用多种循环优化技术, 比如在软件流水前应用循环展开, 提高资源利用率和指令级并行度. 理想情况下, HXDSP 4 个执行簇的 84 个计算部件可以同时进行 84 次运算操作. 显然很多时候单个循环很难充分利用 HXDSP 中 4 个执行簇的功能部件, 即使进行软件流水依然会有很多资源空闲. 循环展开可

① 基金项目: 国家核高基重大专项 (2012ZX01034-001-001)
收稿时间: 2022-03-28; 修改时间: 2022-04-22, 2022-06-01; 采用时间: 2022-06-10; csa 在线出版时间: 2022-08-19

以增加循环体内的代码数量, 可以使 HXDSP 软件流水调度的效果更佳, 提高软件流水的指令并行度. 在软件流水中, 由于多个循环体重叠执行, 增加了对寄存器的需求. HXDSP 每个执行簇中都有大量的寄存器组, 故寄存器压力一般不会成为问题. 循环展开因子和软件流水启动间隔 (initiation interval, II) 是影响 HXDSP 软件流水效果的关键因素[3], 过小的循环展开因子无法充分利用硬件资源, 过大的循环展开因子会造成循环体过长无法成功软件流水. 如何选取合适的循环展开因子和软件流水启动间隔一直是 HXDSP 编译优化所面临的问题.

多任务学习[4] 同时优化多个目标, 相较于单任务学习独立训练和预测单个目标, 多任务学习可以更好地利用多个目标间的相关性提升模型的整体性能和泛化性. 目前在循环程序上的优化技术通常将软件流水和循环展开视为两个不相关的任务进行优化, 这样并不能利用二者之间的联系来生成更高质量的编译指令. 近年来, GNN[5] 逐渐在处理非结构化数据上展现优势, 而程序有着丰富的语法语义关系也更适合表示为图结构数据. 相较于 CNN[6] 和 RNN[7], GNN 更能捕捉程序的语法语义信息. 本文提出一种基于多任务深度学习和 GNN 的方法 MultiOpt, 该方法利用 Clang/LLVM 从源程序中构建抽象语法树 和数据控制流图作为模型输入, 并在从 SPEC, Polybench, NPB 等[8] 基准测试集中提取的 2 000 多个循环中训练模型, 最后预测循环的软件流水启动间隔和循环展开因子. 在 DSP 标准测试集 DSPStone[9] 上, 该多任务方法取得了相较于单任务 12 % 的性能提升.

2 相关研究

经过几十年的发展, 软件流水已经成为一种很成熟和常见的循环优化技术. 软件流水通过将循环的连续多个迭代交替执行, 挖掘跨迭代间指令的并行性. 软件流水中不同迭代的指令可以并行执行, 进而提高了循环的运行效率[10]. 模调度[11] 是一种经典的软件流水算法, 首先通过循环依赖关系和机器资源确定一个软件流水的最小的 II, 然后再由此确定一个合适的流水调度方案. 如果调度不成功则逐渐增加 II 并重复此过程, 直至出现成功的调度或者在 II 增加到某个值时放弃软件流水. 如果直接从 1 开始递增寻找合适的 II, 那将可能会在找到最佳 II 的过程中花费很长时间; 直接选择一个较大的 II 可能会流水调度效果较差, 没有很高的指令并行度, 可见模调度 II 的选取直接影响程序编译效率.

目前基于分簇架构的软件流水研究也有不少, Aleta 等[12] 提出了基于分簇架构的模调度框架 AGAMOS, Llosa 等[13] 提出了同时进行指令分簇和软件流水的摆动模调度算法. 王向前等[3] 研究了国产分簇架构 DSP 的模调度框架, 讨论了循环展开和模调度性能的关系, 并提出了模变量扩展的算法框架. Sanchez 等[14] 提出了基于分簇架构和循环展开的模调度框架, 在循环展开的同时进行指令调度和指令分簇, 不同迭代的指令在不同的执行簇上执行.

随着机器学习以及深度学习在各个领域取得了显著成果, 越来越多的研究人员把机器学习技术应用到了编译优化领域. Stephenson 等[15] 使用监督学习预测循环的最佳展开式因子, 但只考虑了循环展开单个因素对编译效果的影响. Mendis 等[16] 提出了一个基于 GNN 和模仿学习的自动向量化算法, 取得了不错的效果. Cummins 等[17] 利用深度神经网络从源程序中提取特征进行编译器优化因子选择, 对最佳程序执行设备映射和线程粗化因子做了预测, 但这两个任务是独立训练和预测的, 并没有联系起来. Sánchez 等[18] 提出将软件流水启动间隔和循环展开因子映射为二维空间进行搜索的方法, 探究了如何选取合适的循环展开因子和软件流水启动间隔对程序进行编译, 但是采用的是基于迭代编译的方法, 对时间成本没有做出考虑. 程序编译过程中有很多的编译参数和优化因子选项, 只考虑单个因子往往不能获得最优的编译效果. HXDSP 机器上循环程序的软件流水与循环展开是互相影响的, 所以 HXDSP 的软件流水优化应该跟循环展开结合起来才能发挥其硬件资源优势.

多任务深度学习通过在不同任务间共享部分参数来增强模型表示和泛化能力, 按照参数共享机制可分为硬参数共享和软参数共享[19]. 硬参数共享会在网络中会共享一部分网络层, 不同的任务之间共享一些低层次的特征, 而不同的任务在高层拥有自己的网络层. 软参数共享不要求任务之间共享底层的网络层, 通过对底层的网络参数进行正则化的约束来保持任务之间的联系性. 在真实的应用场景中, 需要进行多个任务的学习是很常见的. 例如在自动驾驶系统中, 可能需要同时进行交通目标检测、道路划分和信号灯识别等多个

任务. 对一个 C 程序的编译中, 会有很多的编译优化选项和优化因子的选择.

在传统的编译优化研究中, 研究人员通常只提取程序的单一信息来表示 IR[20–22], 例如数据流、控制流等. 这些 IR 表示方法的缺点是无法全面的保留程序的信息. 关于将程序的数据流和控制流结合在一起得到 CDFG 进行编译优化的研究工作, 目前有 Click 等[23]、Bruan 等[24] 以及 Blindell 等[25] 做过. 但是他们所提出的图表示要么因为添加了大量的节点变得十分臃肿, 要么不满足程序的图表示的一些规则.

3 基于多任务深度学习的优化方法

3.1 系统概述

不同于单任务深度学习独立优化每个目标, 多任务深度学习会同时优化多个目标. 本文的多任务深度学习方法同时优化软件流水启动间隔和循环展开因子两个目标, 也就是模型最后有两个预测值. C 程序的 AST 和 LLVM IR 都可表征程序的语法和语义信息, 其中 AST 与源程序基本相对应, 而 LLVM IR 有与源语言和目标机器的无关性[26]. 本文利用 LLVM/Clang 提取了 AST 和 CDFG 两种 C 程序的图表示作为模型的输入. MultiOpt 网络架构基于门控图神经网络[27], 整个系统框架如图 1 所示.

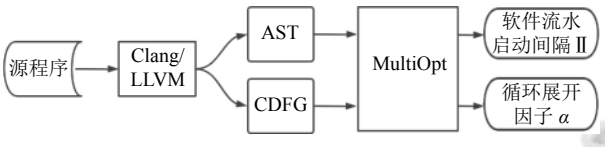

图 1 系统框架

3.2 构建源程序的图表示

本文将源程序转化为了两种图数据表示, 分别是由源程序生成的 AST 和与目标机器无关的 LLVM 中间表达生成的 CDFG. AST 由 LLVM 编译器前端 Clang 从源代码解析出来, 保留了源程序完整的语法结构. CDFG 是由基于静态单赋值 (SSA) [28] 形式的 LLVM 中间表示 IR 组合控制流和数据流生成的. 本文在 SSA 形式的 LLVM IR 的基础上设计了能够全面表示程序信息的 CDFG, 首先将源程序转换为 SSA 形式的 LLVM IR, 在中间 IR 的基础上生成程序的 SSA 图和控制流图 (CFG), 再将 SSA 图和 CFG 结合在一起生成 CDFG. 此时 CDFG 已经包含程序的数据流信息和控制流信息.

3.2.1 构建抽象语法树

抽象语法树是对源程序进行词法分析、语法分析和语义分析后得到的中间表达形式, 它是编译器前端使用上下文无关文法解析后得到的描述代码语法规则和执行顺序的树状数据结构. 树上每一个节点都代表源程序中的某个结构, 叶子节点一般代表源程序中的标识符. 虽然并不是源程序每一个标识符都会在 AST 树上呈现, AST 的生成过程中相应的也会增加或删除一些节点来完善程序的语法语义信息, 但 AST 树基本上和源程序基本是一一对应的. 本文构建的抽象语法树是由在普通 AST 上组合数据流 (AST+DF) 生成的, 实际上包含两个步骤.

(1) 首先使用 LLVM 编译器前端 Clang 通过词法分析, 语法分析以及语义分析从标准化源代码中得到 AST 树.

(2) 在生成的 AST 树上添加数据流信息, 具体做法是在存在使用定义链 (use-def) [29] 的节点之间添加一条 use-def 边 (虚线), 如图 2 所示.

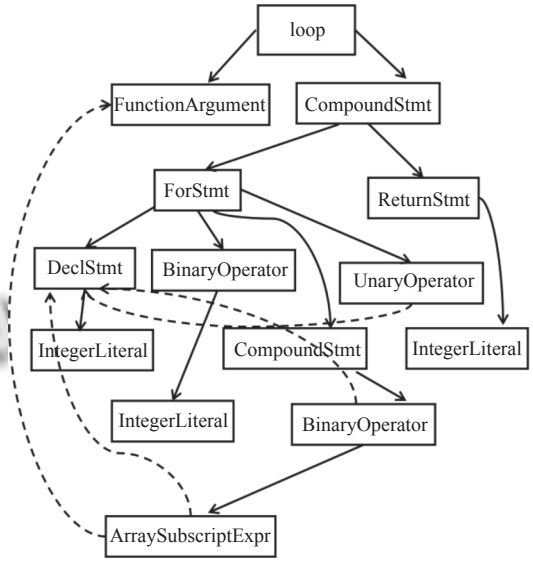

图 2 抽象语法树

3.2.2 构建数据控制流图

数据控制流图是基于 LLVM IR 构建的. LLVM IR 是与机器平台无关的 SSA 形式中间表达, 本文基于 IR 分别构建源程序的 SSA 图和 CFG, 然后再将数据流图和控制流图合成为 CDFG. 主要包含 3 个步骤.

(1) 通过 LLVM 编译前端 Clang 生成源程序的 SSA 形式的中间表达 IR. LLVM IR 是与目标机器无关

的中间表达, IR 中的指令代码采用三地址指令形式, 可以用来进行与目标无关的优化.

(2) 对 IR 生成关于基本块的控制流图, 利用 LLVM 内置 Pass 就可实现. 每一个节点为一个基本块, 各个节点之间的边指向是基本块间的跳转方向, 块内的指令顺序执行.

(3) 如图 3 所示, 为 SSA 形式的 LLVM IR 中的每一条指令在 CDFG 中创建一个节点, 然后根据 CFG 图中的指令执行顺序和基本块跳转方向等控制流信息在节点之间创建控制流边 (实线), 再根据指令之间的数据依赖关系为相关指令节点之间创建数据流边 (虚线).

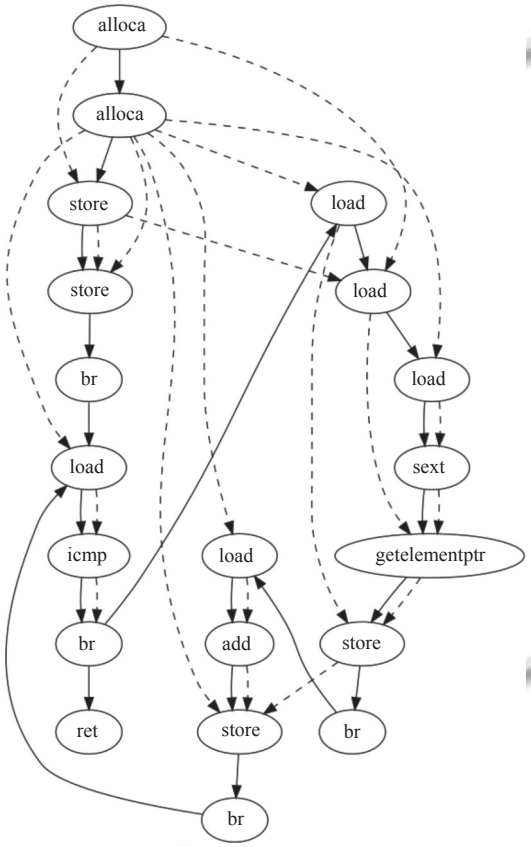

图 3 数据控制流图

3.3 MultiOpt: 多任务图网络

本文提出的多任务深度学习模型 MultiOpt 基于硬参数共享机制, 除预测层外其他网络层参数共享, 图 4 显示了其网络架构.

由 LLVM/Clang 工具链生成的 AST 和 CDFG 包含了完整的源循环程序信息. 本文对 AST 和 CDFG 图中的节点进行独热编码 (one-hot), 每个节点表示为一个 one-hot 向量. 考虑到节点类型可能比较大, 向量维度会很大, 通过一个多层感知器 (MLP) 将节点向量转化为一个固定维度的节点嵌入向量, 如式 (1):

$$h_v = MLP_a(v_t) \tag{1}$$

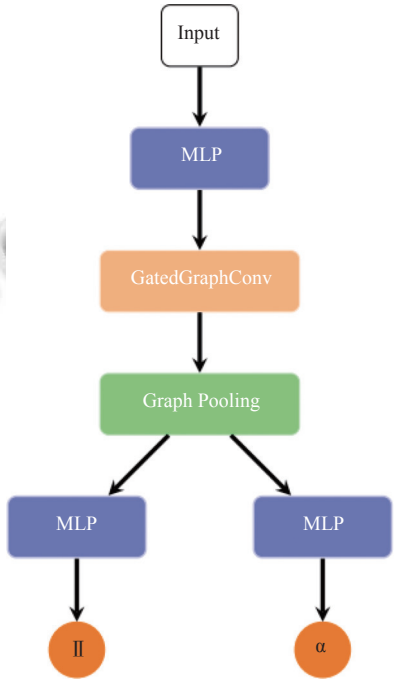

图 4 MultiOpt 网络架构

初始的节点嵌入向量仅包含节点类型信息, 不包任何程序图的结构信息, 无法表达程序图中的边信息和节点连通性. 在传播层中, 每个节点的初始节点嵌入向量都将会经过固定 T 轮消息传递, 生成最终的节点嵌入向量. 在每一轮迭代中, 每个节点根据当前节点嵌入向量和连接的边类型信息生成消息, 并传播此消息给对应边连通的邻居节点. 在消息传播结束后, 每个节点会通过式 (2) 聚合收到的消息. 如式 (3) 所示, 每个节点将根据收到的消息和当前节点嵌入向量来更新自己的节点嵌入向量. 其中f_{prop}是一个门控循环单元 (GRU), GRU 是 RNN 的一种变体, 与长短记忆力网络 (LSTM)[30] 很相似, 但 GRU 参数更少、效果更好.

$$a_v = \sum_{v(u,\theta)\in E} f_{\text{msg}}(h_v, e_t) \tag{2}$$

$$h'_v = f_{\text{prop}}(a_v, h_v) \tag{3}$$

如式 (4) 所示, 在预测层中每张图会聚合图中所有节点嵌入向量得到整个图的嵌入向量 h_G. 其中f_m和

g_m 是全连接神经网络, f_m 中的激活函数使用 Sigmoid, 其最终输出是一个 0, 1 之间的数值, g_m 使用 tanh 函数激活输出. ⊙是哈达玛积, 将两个矩阵对应位置相乘, 它配合 f_m 和 g_m 起到一个注意力机制的效果. 目前为止介绍的所有网络层都是多任务学习中的共享层, 在最后就是两个多层感知器分别输出两个任务的预测值. 如式 (5) 和式 (6) 所示, 这两个 MLP 是完全独立的, 不共享参数.

$$h_G = \sum_{v\in} f_m(h_v) \odot g_m(h_v) \tag{4}$$

$$out_{II} = MLP_M(h_G) \tag{5}$$

$$out_{\partial} = MILP_{\partial}(h_G) \tag{6}$$

3.4 HXDSP 多簇软流水优化

HXDSP 的指令级并行主要通过指令分簇和软件流水来实现. HXDSP 体系结构如图 5 所示, 其拥有 4 个执行簇 (XYZT), 4 个执行簇内部结构完全相同. 循环展开后的循环体内指令数量会倍增, 再进行软件流水可以提高多簇架构处理器的资源利用率和指令级并行度. 如果对循环展开后的循环体指令视作普通指令块一样进行指令分簇和调度, 会无法利用 4 个执行簇的同构性和循环展开指令的相似性. 故本文对循环展开后的循环体内指令做出了相应优化形成了多簇软件流水. 不同迭代间的指令会均分在 4 个执行簇上, 这样 4 个执行簇会形成相似的软件流水且有较好资源负载均衡. 优先形成单指令多数据流 (SIMD) 指令, 并尽可能利用 HXDSP 的单双字寻址模式来同时给 4 个执行簇进行读写, 充分利用 HXDSP 的多数据总线并行读写能力.

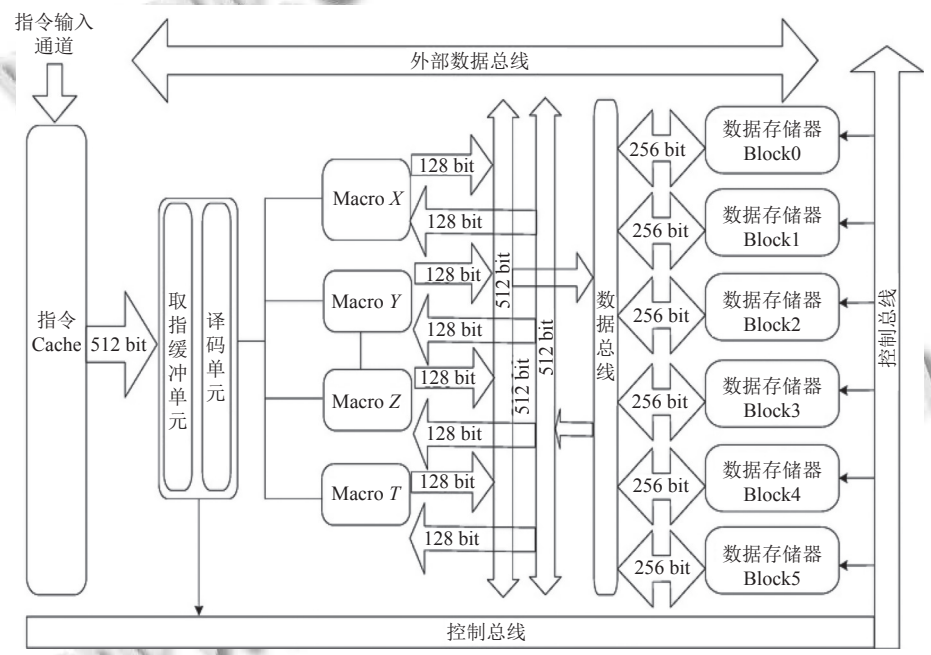

图 5 HXDSP 体系结构

4 实验及结论

4.1 数据集构建

机器学习的触角很早就伸到了编译器领域, 在这个深度学习火热的时代, 深度学习编译器上基于机器学习和搜索的自动调优研究也愈发火热. 但是编译器领域的基准测试集一般只包含十几个例子, 很难满足深度学习模型训练的需求. 本文通过循环存储库 LORE[8], 从 SPEC、Polybench 和 netlib 等基准测试集和公开库提取了 2219 个 C 程序循环用于 MultiOpt 模型的训练. 表 1 显示了各个基准测试集提取出的循环数目. 提取出的循环经过了一定的筛选和处理, 去除了循环嵌套层数超过 5 和循环体内代码过长的循环.每个循环会保留循环体内代码和外部的必要数据引用来形成新的更简洁的循环 C 程序.

4.2 实验环境

本文循环程序编译环境基于 LLVM 10.0, 程序运行硬件平台为 HXDSP1042, 深度学习环境基于 Ubuntu 20.04 和 PyTorch Geometric 2.0. 使用 Adam 优化器来

进行反向传播更新网络模型的参数, 选择标准交叉熵作为损失函数, 对预测软件流水间隔和循环展开因子两个任务同步训练.

表 1 基准测试集的循环数目

测试集	循环数目
ALPbench	63
ASC-llnl	22
Fhourstones	1
FreeBench	56
GitApps	36
Kernels	123
NPB	237
TSVC	153
cortexsuite	94
libraries	311
livermore	73
machinelearing	27
mediabench	427
netlib	47
opencv-kernels	4
polybench	92
scimark2	3
spec2000	214
spec2006	236
总计	2219

4.3 实验结果

为了验证 HXDSP 上软件流水优化使用多任务学习的有效性, 本文设计了单任务学习进行对比实验. 选用 DSP 基准测试集 DSPStone 进行测试, 以基于 LLVM 的 HXDSP 编译器作为基准计算运行加速比.

图 6 是多任务和单任务下的 HXDSP 多簇软件流水的加速比. 多任务是使用 MultiOpt 网络预测的软件流水启动间隔和循环展开因子进行编译, 循环展开后进行模调度. 任务 1 和任务 2 属于单任务学习, 任务 1 是对循环展开因子进行预测, 使用的网络结构和 MultiOpt 类似, 在网络的最后只有一个 MLP, 也只有一个输出. 任务 2 是对软件流水启动间隔进行预测, 网络结构和任务 1 的网络结构相同. 任务 1 以网络预测的循环展开因子进行编译后计算运行加速比, 任务 2 以预测的软件流水启动间隔进编译后计算运行加速比, 其他编译参数任务 1 和任务 2 保持相同. 图 6 展示了 DSPStone 中 3 个典型测试用例在多任务和单任务下的加速比, 从任务 1 和任务 2 可以看出, 基于门控图神经网络的模型预测的循环展开因子和软件流水启动间隔都优于基于 LLVM 的 HXDSP 编译器自动循环展开和软件流水. 在 3 个测试用例中, 多任务 MultiOpt 的加速比均高于任务 1 和任务 2, 取得了相对于单任务至少 12% 的性能提升.

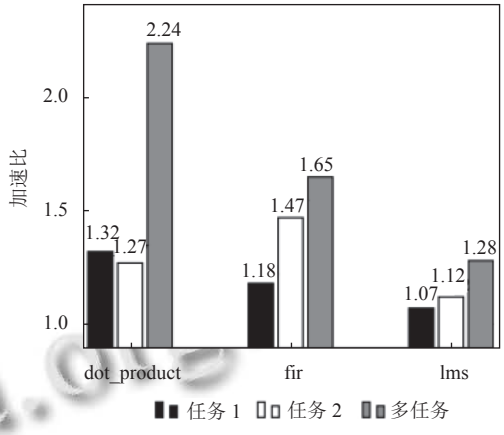

图 6 多任务与单任务对比实验

MultiOpt 基于 GGNN 网络并在图特征聚合时使用了注意力机制. 本文选取了 LSTM、GCN[31] 和 GraphSAGE[32] 三个模型与 MultiOpt 进行多任务深度学习对比实验. 其中 LSTM 是循环神经网络的一种, GCN 和 GraphSAGE 都属于图神经网络.

表 2 展示了不同模型在 DPStone 测试用例上的加速比. 可以看出基于图神经网络的模型都优于 LSTM 模型, MultiOpt 也优于同属于 GNN 的 GCN 和 GraphSAGE. MultiOpt 相较于第二的模型 GCN 也提高了 9% 的性能. 不同于序列模型 LSTM, 图神经网络不简单把程序视为一串线性符号, 它可以学习到程序图的结构性信息. MultiOpt 的门控循环单位在聚合邻居时更有优势, 图特征聚合时使用的注意力机制也能对不同节点给与不同的权重, 这都可以能更好地表示程序图结构信息.

表 2 模型对比实验

模型	加速比
LSTM	1.28
GCN	1.57
GraphSAGE	1.41
MultiOpt	1.71

图 7 显示了 MultiOpt 在消息传播层的嵌入层数从 1 增加到 8 的过程中模型性能变化. 嵌入层数决定了消息传播和邻居聚合会进行几个轮次, 决定每个节点自己的消息会辐射到的邻居范围和自己能感知到的消息范围. 随着嵌入层数从 1 增加到 4, 模型的性能逐渐提高, 在嵌入层数为 4 时取得最佳性能. 嵌入层数的

增加会增加消息传播的轮次, 每个节点的消息会传播到更远范围的节点, 这样最后的嵌入向量会学习到更丰富的图结构信息. 当嵌入层数增大到 5 及以上时, 模型的性能出现了下滑. 嵌入层数过大时, 会出现所有的节点嵌入向量趋于收敛于同一个值. 这样整个模型的表达能力就会变弱.

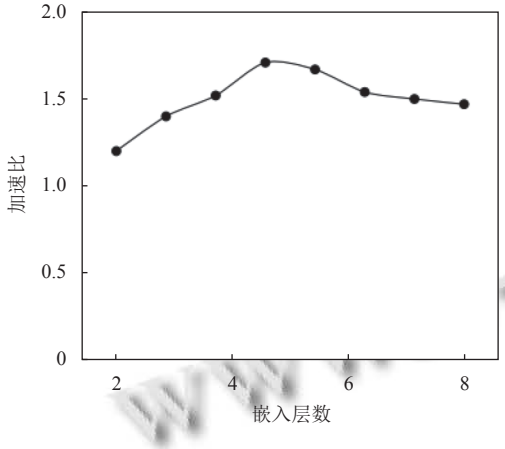

图 7 嵌入层数的影响

5 结论与展望

针对目前 HXDSP 编译优化普遍将软件流水和循环展开分成两个任务进行优化而无法利用多个任务间的相关性提升模型性能的问题, 本文提出一种多任务深度学习模型 MultiOpt, 该模型将 HXDSP 平台上 C 程序的软件流水和循环展开结合在一起优化. MultiOpt 以 C 程序的 AST 和 CDFG 作为输入, 同时以软件流水启动间隔和循环展开因子两个任务进行训练, 并与单任务学习进行对比. 实验表明, 多任务学习取得了相对于单任务 12% 的加速效果, MultiOpt 模型也优于其他网络模型. 但是, 本文仅选取了 HXDSP 编译优化中的两个任务, 未考虑更多任务下的模型效果. 在未来的工作中, 可以考虑更多的 HXDSP 编译优化项目并改进模型以容纳更多的学习任务.

参考文献

1 CET38. HXDSP 软件用户手册. 合肥: 中国电子科技集团第三十八研究所, 2017. 7–8.
2 Weiss S, Smith JE. A study of scalar compilation techniques for pipelined supercomputers. ACM SIGOPS Operating Systems Review, 1987, 21(4): 105–109. [doi: 10.1145/36204.36191]
3 王向前, 郑启龙, 洪一. 分簇结构模调度框架研究. 中国科学技术大学学报, 2016, 46(2): 104–112.
4 Zhang Y, Yang Q. An overview of multi-task learning. National Science Review, 2018, 5(1): 30–43. [doi: 10.1093/nsr/nwx105]
5 Zhou J, Cui GQ, Hu SD, *et al*. Graph neural networks: A review of methods and applications. AI Open, 2020, 1: 57–81. [doi: 10.1016/j.aiopen.2021.01.001]
6 Khan A, Sohail A, Zahoora U, *et al*. A survey of the recent architectures of deep convolutional neural networks. Artificial Intelligence Review, 2020, 53(8): 5455–5516. [doi: 10.1007/s10462-020-09825-6]
7 Yu Y, Si XS, Hu CH, *et al*. A review of recurrent neural networks: LSTM cells and network architectures. Neural Computation, 2019, 31(7): 1235–1270. [doi: 10.1162/neco_a_01199]
8 Chen Z, Gong ZXW, Szaday JJ, *et al*. LORE: A loop repository for the evaluation of compilers. 2017 IEEE International Symposium on Workload Characterization (IISWC). Seattle: IEEE, 2017. 219–228.
9 Zivojnovic V. DSPstone: A DSP-oriented benchmarking methodology. Proceedings of Signal Processing Applications & Technology. Dallas, 1994. 715–720.
10 Sittel P, Kumm M, Oppermann J, *et al*. ILP-based modulo scheduling and binding for register minimization. Proceedings of the 28th International Conference on Field Programmable Logic and Applications. Dublin: IEEE, 2018. 265–266 [doi: 10.1109/FPL.2018.00053]
11 Sittel P, Wickerson J, Kuimm M, *et al*. Modulo scheduling with rational initiation intervals in custom hardware design. Proceedings of the 25th Asia and South Pacific Design Automation Conference. Beijing: IEEE, 2020. 568–573.
12 Aleta A, Codina JM, Sanchez J, *et al*. AGAMOS: A graph-based approach to modulo scheduling for clustered microarchitectures. IEEE Transactions on Computers, 2009, 58(6): 770–783. [doi: 10.1109/TC.2009.32]
13 Llosa J, Gonzalez A, Ayguade E, *et al*. Swing module scheduling: A lifetime-sensitive approach. Proceedings of the 1996 Conference on Parallel Architectures and Compilation Technique. Boston: IEEE, 1996. 80–86. [doi: 10.1109/PACT.1996.554030]
14 Sanchez J, Gonzalez A. Modulo scheduling for a fully-distributed clustered VLIW architecture. Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture. Monterey: IEEE, 2000. 124–133.
15 Stephenson M, Amarasinghe S. Predicting unroll factors

using supervised classification. International Symposium on Code Generation and Optimization. New York: IEEE, 2005. 123–134. [doi: 10.1109/CGO.2005.29]

16 Mendis SC, Yang C, Pu YW, *et al*. Compiler auto-vectorization with imitation learning. Proceedings of the 33rd International Conference on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 1310.

17 Cummins C, Petoumenos P, Wang Z, *et al*. End-to-end deep learning of optimization heuristics. Proceedings of the 26th International Conference on Parallel Architectures and Compilation Techniques (PACT). Portland: IEEE, 2017. 219–232. [doi: 10.1109/PACT.2017.24]

18 Sánchez F, Cortadella J, Badia RM. Optimal exploration of the unrolling degree for software pipelining. Journal of Systems Architecture, 1999, 45(6–7): 505–517. [doi: 10.1016/S1383-7621(98)00020-4]

19 Bakker B, Heskes T. Task clustering and gating for Bayesian multitask learning. The Journal of Machine Learning Research, 2003, 4: 83–99.

20 Ebner D, Brandner F, Scholz B, *et al*. Generalized instruction selection using *SSA*-graphs. Proceedings of the 2008 ACM SIGPLAN-SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems. Tucson: ACM, 2008. 31–40. [doi: 10.1145/1375657.1375663]

21 Cytron R, Ferrante J, Rosen BK, *et al*. Efficiently computing static single assignment form and the control dependence graph. ACM Transactions on Programming Languages and Systems, 1991, 13(4): 451–490. [doi: 10.1145/115372.115320]

22 Wang SS, Xiao CL, Liu WJ, *et al*. A comparison of heuristic algorithms for custom instruction selection. Microprocessors and Microsystems, 2016, 45: 176–186. [doi: 10.1016/j.micpro.2016.05.001]

23 Click C, Paleczny M. A simple graph-based intermediate representation. ACM Sigplan Notices, 1993, 30(3): 35–49. [doi: 10.1145/202530.202534]

24 Braun M, Buchwald S, Zwinkau A. FIRM: A Graph-based Intermediate Representation. Karlsruhe: KIT, 2011.

25 Blindell HB, Lozano CL, Carlsson M, *et al*. Modeling universal instruction selection. Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming. Cork: Springer, 2015. 609–626.

26 Lattner C, Adve V. LLVM: A compilation framework for lifelong program analysis & transformation. Proceedings of 2004 International Symposium on Code Generation and Optimization. San Jose: IEEE, 2004. 75–86. [doi: 10.1109/CGO.2004.1281665]

27 Li YJ, Tarlow D, Brockschmidt M, *et al*. Gated graph sequence neural networks. Proceedings of the 4th International Conference on Learning Representations. San Juan: ICLR, 2016.

28 Bilardi G, Pingali K. Algorithms for computing the static single assignment form. Journal of the ACM, 2003, 50(3): 375–425. [doi: 10.1145/765568.765573]

29 Pande HD, Landi WA, Ryder BG. Interprocedural def-use associations for C systems with single level pointers. IEEE Transactions on Software Engineering, 1994, 20(5): 385–403. [doi: 10.1109/32.286418]

30 Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735–1780. [doi: 10.1162/neco.1997.9.8.1735]

31 Schlichtkrull M, Kipf TN, Bloem P, *et al*. Modeling relational data with graph convolutional networks. Proceedings of the 15th International Conference on the Semantic Web. Heraklion: Springer, 2018. 593–607.

32 Hamilton WL, Ying R, Leskovec J. Inductive representation learning on large graphs. Proceedings of the 31st International Conference on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 1025–1035.

(校对责编: 牛欣悦)