计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
62(1): 22−57, 2025

面向处理器微架构设计空间探索的加速方法综述

王 铎[1,2] 刘景磊[3] 严明玉[1,2] 滕亦涵[1,2] 韩登科[1,2] 叶笑春[1,2] 范东睿[1,2]

[1](处理器芯片全国重点实验室(中国科学院计算技术研究所) 北京 100190)

[2](中国科学院大学计算机科学与技术学院 北京 100049)

[3](中国移动研究院 北京 100053)

(wangduo@ict.ac.cn)

Acceleration Methods for Processor Microarchitecture Design Space Exploration: A Survey

Wang Duo[1,2], Liu Jinglei[3], Yan Mingyu[1,2], Teng Yihan[1,2], Han Dengke[1,2], Ye Xiaochun[1,2], and Fan Dongrui[1,2]

[1](*State Key Lab of Processors* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)

[2](*School of Computer Science and Technology*, *University of Chinese Academy of Sciences*, *Beijing* 100049)

[3](*China Mobile Communications Research Institute*, *Beijing* 100053)

Abstract Central processing unit is the most important computing infrastructure nowadays. To maximize the profit, architects design the processor microarchitecture by trading-off multiple objectives including performance, power, and area. However, because of the tremendous instructions of workloads running on the processors, the evaluation of individual microarchitecture design point costs minutes to hours. Furthermore, the design space of the microarchitecture is huge, which results that the exploration of comprehensive design space is unrealistic. Therefore, many machine-learning-assisted design space exploration acceleration methods are proposed to reduce the size of evaluated design space or accelerate the evaluation of a design point. However, a comprehensive survey summarizing and systematically classifying recent acceleration methods is missing. This survey paper systematically summarizes and classifies the five kinds of acceleration methods for the design space exploration of the processor microarchitecture, including the workload selection of software design space, the partial simulation of workload instructions, the design point selection, the simulation tools, and the performance models. This paper systematically compares the similarities and differences between papers in the acceleration methods, and covers the complete exploration process from the software workload selection to the hardware microarchitecture design. Finally, the research direction is summarized, and the future development trend is discussed.

Key words processor microarchitecture design; design space exploration; performance model; workload selection; software simulation

摘 要 中央处理器是目前最重要的算力基础设施.为了最大化收益,架构师在设计处理器微架构时需要

收稿日期: 2023-05-07; **修回日期**: 2024-01-15

基金项目: 国家自然科学基金项目(62202451); 中国科学院国际伙伴计划项目(171111KYSB20200002); 中国科学院稳定支持基础研究领域青年团队计划项目(YSBR-029); 中国科学院青年创新促进会项目(Y2021039); 中科院计算所-中国移动研究院联合创新平台项目

This work was supported by the National Natural Science Foundation of China (62202451), the International Partnership Program of Chinese Academy of Sciences (171111KYSB20200002), the CAS Project for Young Scientists in Basic Research (YSBR-029), the CAS Project for Youth Innovation Promotion Association (Y2021039), and the Institute of Computing Technology, Chinese Academy of Sciences-China Mobile Communications Group Co., Ltd. Joint Institute.

通信作者: 严明玉(yanmingyu@ict.ac.cn)

权衡性能、功耗、面积等多个目标. 但处理器运行负载的指令多，单个微架构设计点的评估耗时从 10 min 到数十小时不等. 加之微架构设计空间巨大，全设计空间暴力搜索难以实现. 近些年来许多机器学习辅助的设计空间探索加速方法被提出，以减少需要探索的设计空间或加速设计点的评估，但缺少对加速方法的全面调研和系统分类的综述. 对处理器微架构设计空间探索的加速方法进行系统总结及分类，包含软件设计空间的负载选择、负载指令的部分模拟、设计点选择、模拟工具、性能模型 5 类加速方法. 对比了各加速方法内文献的异同，覆盖了从软件选择到硬件设计的完整探索流程. 最后对该领域的前沿研究方向进行了总结，并放眼于未来的发展趋势.

关键词 处理器微架构设计；设计空间探索；性能模型；负载选择；软件模拟

中图法分类号 TP302

DOI: 10.7544/issn1000-1239.202330348 **CSTR:** 32373.14.issn1000-1239.202330348

通用处理器是目前重要的算力基础设施，CPU 是其中最核心的部件. CPU 微架构（microarchitecture）设计需要面对各类场景，在性能、功耗、面积、成本等多个指标上进行权衡才能满足市场需求[1-2]，因此需要架构师进行微架构设计空间探索（design space exploration, DSE）. 微架构设计空间探索的工作包含对性能指标的预测[3-12]、寻找多目标约束下的最优解[1-2,13-14]、评估关键参数的改变对性能指标的影响[2,12,15-16] 等.

微架构设计空间探索的效果决定处理器的市场竞争力，但探索过程复杂且耗时，对设计人员提出了极高的挑战[17]. 首先处理器评估时运行的负载多且耗时. 例如常用的通用处理器性能测试套件 SPEC CPU 2017[18] 包含 23 个人工智能、科学计算等不同领域的负载，每个负载达到数万亿条指令，在物理机上完成 1 次运行需要若干小时. 其次，微架构设计涉及的电路模块众多且复杂，基于寄存器传输级（register transfer level, RTL）电路的设计与评估耗时，其模拟速度比物理机慢数个量级，需要花费月级别的时间才能完成 1 个负载的完整评估. 面对日益复杂的微架构设计与随之指数级增加的设计空间，全空间暴力探索无法实现[12]. 因此处理器微架构设计空间探索的加速方法变得越来越重要[12,19-21]，吸引了众多研究者.

然而，针对处理器微架构设计空间探索的加速方法，现有的综述缺乏从完整处理器微架构设计空间探索的范围对加速方法进行系统性分类和对比，并且随着设计空间的扩展和机器学习的发展，近几年出现了新的加速方法. 现有的综述在早期的统计模拟和性能模型等方面完成调研，如 Eeckhout 等人[12] 在 2004 年总结了负载选择和部分模拟的早期工作，Yi 等人[19] 在 2006 年总结了模拟工具和负载选择等工作. Guo 等人[20] 总结了统计模拟和加速方法，从减少来源的角度对加速方法进行了分类. O’Neal 等人[21] 总结了部分 CPU/现场可编程逻辑阵列/图形处理器的性能和功耗预测模型，以预测模型替代耗时的评估来加速探索过程. 但 O’Neal 等人[21] 仅对特定阶段的加速方法进行总结，缺乏如设计点选择加速方法的分类及对比. 因此为了填补该领域研究的缺失，本文系统性分类并总结对比了处理器微架构设计空间探索的加速方法.

本文对处理器微架构设计空间探索的加速方法进行全面总结和系统性分类，提出一种按所加速的探索阶段进行分类的方法，包括负载选择、部分模拟、设计点选择、模拟工具、性能模型，并对比了各加速方法内文献的异同. 此外，本文分析了处理器微架构设计空间探索加速方法的新趋势，帮助研究者寻找重要挑战和潜在的研究方向.

1 背　　景

本节首先介绍处理器微架构设计空间探索的相关概念，其次介绍探索加速方法中常用的组合优化方法与机器学习模型.

1.1 处理器微架构设计空间探索概念

本节将主要介绍处理器微架构设计空间探索的基础定义及其形式化表示.

本文主要关注处理器微架构的设计空间，包括根据具体处理器微架构参数形成的硬件设计空间和根据要运行的负载及其输入集所形成的软件设计空间. 本文涉及到的负载面向通用嵌入式/桌面/服务器处理器，优化性能指标包括执行时间、功耗等. 所涉及到的微架构设计涵盖通用单核处理器微架构、同时多线程（simultaneous multithreading, SMT）架构、同构/异构多核处理器（homogeneous/heterogeneous multi-core

processor)、超长指令集架构(very long instruction word, VLIW)、片上多处理器(chip multiprocessor, CMP). 涉及的设计平台包括应用专用集成电路(application specific integrated circuit, ASIC)和现场可编程逻辑阵列(field programmable gate array, FPGA). 处理器微架构设计中常见的组件缩写如下: 高速缓存(cache)、指令高速缓存(instruction cache, Icache)、数据高速缓存(data cache, Dcache)、分支目标缓冲(branch target buffer, BTB)、转址旁路缓冲(translation-lookaside buffer, TLB)、分支预测器(branch predictor, BP).

定义 1. 设计空间. 为了形式化表示, 不同的微架构参数被组合为不同的特征向量$\boldsymbol{x}$, 而每一个$\boldsymbol{x}$都被称为一个设计点. 所有不同的设计点共同构成了完整的硬件设计空间D.

定义 2. 性能指标. 不同的设计点$\boldsymbol{x}$对应不同的性能结果. 为了评估不同设计点$\boldsymbol{x}$的性能指标y, 我们形式化地将其表示为$y=f(\boldsymbol{x})$的形式. 处理器设计空间探索中常关注的性能指标[9-11,22-23]包括执行时间(execution time)、每条指令的周期数(cycle per instruction, CPI)、每周期的指令数(instruction per cycle, IPC)、能耗(energy)、功率(power)、面积(area), 复合型性能指标包括能耗延迟积(energy delay product, EDP)等. 当一个设计点$\boldsymbol{x}$通过评估工具得到性能指标后, 我们就称得到了一个样本(sample).

微架构设计空间探索的工作包含对性能指标的预测、寻找多目标约束下的最优解、评估关键参数的改变对性能指标的影响等. 单目标设计空间探索中, 优化目标常为最小化性能指标的预测误差, 如平均绝对误差[9-11]、平均相对误差[3-8], 或最大化真实值与预测值的相关性, 如皮尔森相关因子[10-11,23]. 在多目标设计空间探索中, 目标微架构需要平衡多个性能指标, 优化目标常为最小化与帕累托最优解集的距离[1-2,13-14]. 此外, 设计空间探索的工作可评估关键参数的改变对执行时间、能耗、温度、面积等性能指标的影响[2,15-16], 帮助架构师进行权衡和参数选择.

定义 3. 帕累托最优解集. 对于多目标设计空间的探索, 最重要的任务之一是找到帕累托最优解集(Pareto optimal set)[2,14,24]. 给定一个n个目标的最小化问题, 如果

$$\begin{aligned}&\forall i\in[1,n],\ f_i(\boldsymbol{x}^*)\leqslant f_i(\boldsymbol{x}),\\&\exists j\in[1,n],\ f_j(\boldsymbol{x}^*)<f_j(\boldsymbol{x}),\end{aligned}\tag{1}$$

则$\boldsymbol{x}$被$\boldsymbol{x}^*$所支配, 即$\boldsymbol{x}^*\succcurlyeq\boldsymbol{x}$, 否则$\boldsymbol{x}$不被$\boldsymbol{x}^*$所支配, 即$\boldsymbol{x}^*\not\succcurlyeq\boldsymbol{x}$. 在所有的设计点中, 没有被其他设计点支配的一组设计点称为帕累托最优解集Ω. 我们将这个集合形式化为

$$\Omega=\{\boldsymbol{x}|\boldsymbol{x}^*\not\succcurlyeq\boldsymbol{x},\forall\boldsymbol{x}^*\in D\}.\tag{2}$$

帕累托最优解集的目标值集构成帕累托最佳边界. 已知的设计点$\boldsymbol{x}$的非支配级别是对$\boldsymbol{x}$被嵌套到次优(非帕累托)解中深度的度量[25-26].

1.2 组合优化与机器学习

本节对设计空间探索加速方法中所用到的常见组合优化算法和机器学习模型进行简要介绍.

1)组合优化

组合优化问题是整数规划的子集, 通过数学方法去寻找离散变量的最优编排、分组、次序或筛选等. 遗传算法和模拟退火作为仿生物和物理机制的方法常用于解决组合优化问题.

遗传算法(genetic algorithm, GA), 又称进化算法(evolution algorithm, EA), 将信息编码为基因形式. 首先生成初始族群. 对每一代族群, 基于适应度函数评估个体, 通过选择、交叉、变异等操作生成下一代族群(自然选择), 根据需求迭代多代, 最终生成最优的个体或族群. 在应用于设计空间探索时, 通常将微架构参数表示为一个向量(即基因), 通过交叉 2 个向量、随机改变向量位等操作, 适应度函数常使用微架构的性能指标.

模拟退火(simulated annealing, SA)将固体内能建模为目标函数值, 将温度建模为控制参数, 从初始解开始从邻域中产生新解, 按概率在一定范围内接受使目标函数恶化的解, 循环进行“产生新解并计算目标函数差—判断是否接受新解—接受或舍弃”的迭代过程. 经过大量的迭代, 可以求得优化问题的相对最优解. 然后减小控制参数的值, 重复执行上述迭代过程. 当控制参数逐渐减小并趋于 0 时, 系统也趋于平衡状态, 最后系统状态对应于优化问题的全局最优解. 在应用于设计空间探索时, 将微架构参数表示为一个向量的解, 新解可通过领域选择、随机改变向量位等操作产生, 系统状态常使用微架构的性能指标来评估.

2)机器学习

机器学习方法的关键是基于数据学习变量之间的关系. 具体而言, 机器学习方法可以分为无监督学习和有监督学习 2 类.

学习无监督学习是指在数据无标签的情况下进行学习, 包括聚类算法和降维方法等. 聚类算法是一种基于相似性度量的无监督学习算法. 典型算法包括k均值聚类算法(k-means)、层次聚类(hierarchical clustering)算法. k均值聚类是一种迭代求解算法, 其

将数据分为k组，随机选取k个对象作为初始的聚类中心，然后计算每个对象与各个聚类中心之间的距离，把每个对象分配给距离它最近的聚类中心. 每分配一个对象，重新计算聚类中心. 层次聚类通过自上向下将大类别进行分割，或由下向上对小类别进行聚合，不断重复直到满足某个终止条件. 聚类算法常配合贝叶斯信息准则等确定最佳的聚类数. 主成分分析(principal components analysis, PCA)是一种特征降维方法，通过构造新的、不相关的统计随机变量(称为主成分)来消除变量之间的内在相关性. 主成分分析方法常与层次聚类组合使用(合并简称为PCA-H). 在应用于设计空间探索时，无监督学习常将微架构相关及无关特征表示为一维向量进行学习，聚类算法和降维方法常用于输出代表性的负载集，降维方法也用于构造代表性的特征映射，辅助后续设计空间探索加速.

有监督学习是指在有标签数据的训练集下进行学习，预测测试集的数据标签. 应用有监督学习时，常将微架构参数表示为1维向量作为数据特征，将性能评估指标作为数据标签，利用模拟评估工具获取数据组成训练集.

多项式线性回归(linear regression, LR)试图拟合一个N元M次线性方程，常用最小二乘法优化损失函数. 套索回归(lasso regression)增加惩罚函数进而压缩一些回归系数. 样条回归(spline regression)使用多段多项式线性回归对样本数据进行拟合，并保证这些曲线的接口处也是平滑的.

基于核函数的方法中，支持向量机(support vector machine, SVM)将输入空间投影到高维特征空间上，试图最小化ϵ不敏感损失函数，忽略观测值的ϵ不敏感区域内的错误，并只测量位于该区域之外的错误. 常用核函数包括径向基函数. 高斯过程(Gaussian process, GP)假设训练数据来自一个多元高斯分布，均值向量和协方差矩阵通过一个用户定义的正定核函数获得. 训练过程包括通过最大化对数边际似然值来优化核的超参数. 克里金法(Kriging)同样是依据协方差函数对随机过程进行空间建模和预测(插值)的回归算法.

人工神经网络(artificial neural network, ANN)，简称神经网络. 其以分层方式组织网络

$$\boldsymbol{p}^0(\boldsymbol{x})=\boldsymbol{x},\boldsymbol{p}^{layer+1}(\boldsymbol{x})=\psi\left(\boldsymbol{w}\boldsymbol{p}^{layer}(\boldsymbol{x})+\boldsymbol{b}\right), \quad (3)$$

每一层$layer$可包含一个非线性激活函数ψ，训练点通过输入层呈现给网络，层内通过一个加权变量$\boldsymbol{w}$进行计算. 最后一个隐藏层被连接到一个输出层，获得预测的输出. 训练阶段包括更新网络中连接的权重，以最小化测量预测误差的损失函数.

决策树(decision tree, DT)将知识表示为一个树，每个节点代表一个基于特征和阈值的特定分支. 模型树(model tree)同样将知识表示为树，回归树和模型树的区别就是回归树的叶节点是一个常数值，而模型树的叶节点是分段线性函数.

集成学习是一种常用的用于提升模型效果的优化方法，其训练多个模型(通常称为弱学习器)解决相同的问题，并将它们结合起来以获得更好的结果. 常用的集成方法包括装袋(bagging)法、提升(boosting)法、堆叠(stacking)法. 装袋法相互独立地并行学习这些弱学习器，并将它们组合起来取平均值作为最终输出. 提升法以一种高度自适应的方法，按顺序学习这些弱学习器，并按照某种确定性的策略将它们组合起来. 堆叠法考虑的是异质弱学习器并行地学习，并通过一个元模型将它们组合起来，根据不同弱学习器的预测结果输出一个最终的预测结果. 集成决策树模型包括采用装袋法的随机森林(random forest, RF)和采用提升法的梯度提升回归树(gradient boosting regression tree, GBRT)等.

2 设计空间探索加速方法的分类

本节中，我们将处理器微架构设计空间探索加速方法分为5个类型，覆盖从软件空间负载选择到硬件微架构设计空间设计点采样和模拟的完整探索流程.

首先，设计空间探索过程复杂且耗时，若采用全空间探索的方法，总时间可形式化为

总时间 = 负载数量×指令数量×设计点数量×
单位指令耗时，

即负载数量、指令数量、设计点数量、每个设计点评估的单位指令耗时的乘积. 设计点(即微架构参数的组合)组成处理器微架构硬件设计空间，负载和指令的选择组成软件设计空间. 由于阶乘效应，全空间暴力探索极为耗时，甚至不可接受.

因此，目前的工作主要通过减少设计空间大小和加速设计点评估以加速探索，加速后的探索流程如图1所示. 首先对软件设计空间进行负载选择，再对所选负载的指令序列进行缩减，仅对部分指令片段进行后续运行. 然后，根据硬件设计空间所包含的微架构参数，配合负载特征进行设计点的选择. 期间使用模拟工具运行所选的指令片段，加速设计点评

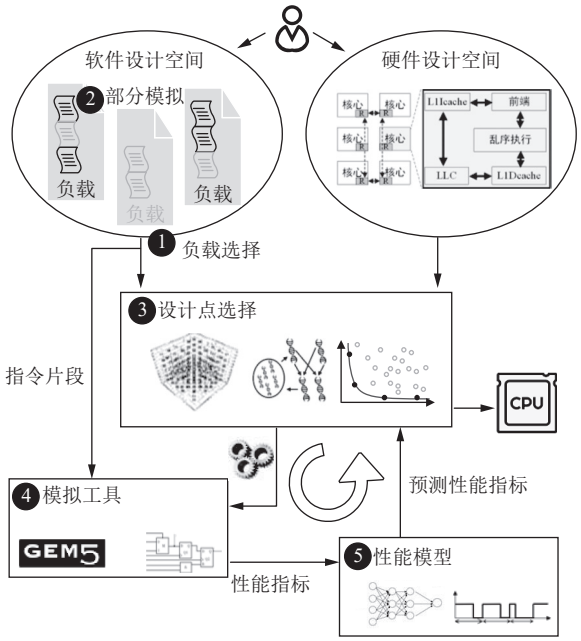

Fig. 1 Design space exploration after acceleration

图 1 加速后的设计空间探索

估, 获得性能指标. 随后, 利用已评估过的设计点训练或构造性能模型, 预测未评估过的设计点的性能指标. 迭代设计空间探索过程, 直到满足架构师需求, 最终生成 CPU 微架构设计. 具体而言, 我们将处理器微架构设计空间探索的加速方法分为表 1 的 5 种类型(按探索顺序先后), 并根据特性进行子类型分类.

1)负载选择. 分析软件空间负载集的执行特征, 通过特征降维和聚类等方法选择具备代表性的子负载和输入集, 减少需要评估的负载数量, 按照选择时依据的特征可分为 3 类, 分别是基于微架构相关特征的方法、基于微架构无关特征的方法、基于微架构相关与无关特征的方法.

2)部分模拟. 分析负载的执行特征, 选择或生成负载的部分指令片段进行后续评估, 缩短负载的评估耗时, 加速设计点的评估, 按照是否采用原始指令片段部分模拟可分为 2 类, 统计采样模拟选取部分指令片段, 而综合模拟不采用原始指令, 生成综合指令踪迹.

3)设计点选择. 面对硬件微架构设计空间, 选择需要评估的设计点, 优化性能指标或在多目标约束下寻找最优解等, 按照选择设计点的方式分为 2 类, 一次性生成的采样方法和迭代搜索方法.

4)模拟工具. 加速单个设计点的评估时间, 快速而准确地获取设计点的性能指标, 按照模拟的抽象级别高低分为 3 类, 高抽象级别的软件模拟、中抽象级别的硬件模拟和低抽象级别的电路设计敏捷开发.

5)性能模型. 建立性能模型, 加速耗时的评估过程, 用已有知识预测未评估设计点的性能指标, 以加速设计空间探索. 按照是否建模微架构组件分为 2 类: 不建模微架构组件的预测模型和建模微架构组件的机械模型.

通过这 5 类加速方法, 处理器微架构设计空间探索总时间减少为

$$\begin{aligned}\text{总时间} = {} & \text{代表性负载数量} \times \text{缩减指令数量} \times \\ & \text{采样设计点数量} \times \text{模拟时间} + \\ & \text{模型训练及预测时间},\end{aligned}$$

相比原始的全空间探索加速了若干数量级.

表 2 对比各加速方法在典型文献中的加速比和

Table 1 Category of Acceleration Methods for Processor Microarchitecture Design Space Exploration

表 1 处理器微架构设计空间探索的加速方法分类

类型	子类型	典型方法
负载选择	基于微架构相关特征的方法	文献 [15, 27]
	基于微架构无关特征的方法	MinneSPEC[28]、文献 [29–32]、BenchSubset[33]、CASH[34]
	基于微架构相关与无关特征的方法	文献 [29, 35–37]、BenchPrime[38]
部分模拟	统计采样模拟	采样单线程[39-43]、采样多线程[44-48]、采样访存[49-51]
	综合模拟	综合单线程[52-55]、综合多线程[56-58]、综合访存[59-62]
设计点选择	采样方法	基于参数敏感度的方法[15,63-67]、基于实验设计的方法[6,25,34,67]
	迭代搜索方法	启发式方法[68-70]、组合优化方法[68,71-74]、统计推理方法[14,25-26,67,75]
模拟工具	软件模拟	SimpleScalar[76], SESC[77], gem5[78]
	硬件模拟	FAST[79], PROTOFLEX[80-81], RAMP Gold[82], HAsim[83], FireSim[84]
	敏捷开发	基于低级语言的平台[85-87]、基于高级语言的平台[50,88-89]
性能模型	特定负载预测模型	参数化模型[2,4,90]、核函数模型[13,68,91]、神经网络模型[3,92-93]、树模型[94-96]、集成学习模型[67,97-98]
	跨负载预测模型	基于负载特征[8,99-100]、基于硬件响应[9,23,101]、基于迁移学习[7,10-11]
	机械模型	分析模型[102-103]、区间模型[104-106]、图模型[107-109]、概率统计模型[110-112]、混合模型[113-115]

准确率. 值得注意的是，加速方法之间是正交关系，因此在完整的设计空间探索流程中，常同时使用若干加速方法以实现叠加效果. 负载选择方法仅选择少量代表负载进行后续评估，如文献 [37] 从 43 个负载中选择 12 个. 这类方法的优势在于算法简单且选择过程快，但加速比较少，且准确率随负载的选择波动较大. 部分模拟方法针对单个负载选择或生成部分指令片段，如 PerfProx[116] 将原始约 1 万亿条指令的负载用 20 亿条指令的代理负载替代，加速效果比负载选择快 2 个数量级，但算法较为复杂，常需要对负载进行细致的分析. 设计点选择方法常常配合性能模型方法使用.

Table 2 Comparison of Acceleration Methods

表 2 加速方法对比

类型	典型方法	加速比	准确率/%
负载选择	文献 [37]	5.2	93.0
部分模拟	文献 [116]	520	94.9
设计点选择	文献 [117]	23 000	99.0
模拟工具	文献 [76]	1 000	95.0
性能模型	文献 [118]	180 000	98.2

相关工作中常使用设计点选择方法决定初始设计点集，再配合性能模型的估计结果进行迭代探索，加速效果来自于只模拟设计空间中少量且有代表性的设计点. 模拟工具中最常用的软件模拟器，如 SimpleScalar[76]，优势在于相比 1~5 kHz 的 RTL 级仿真，软件模拟器的速度可达 1~10 MHz[119]，且灵活性高，但缺点在于需要进行耗时且费力的性能校准，性能误差范围很大. 性能模型通过训练或白盒构造处理器模型，由于仅通过少量公式即可估计性能指标，替代了耗时的软件或硬件模拟，加速效果显著，如 ELSE[118] 在 20 亿个设计点的设计空间中仅详细模拟了 3 000 个设计点，剩余设计点通过模型快速预测，平均 CPI 估计误差仅为 1.8%，然而，缺点在于需要针对特定的处理器设计空间进行建模，目前灵活性和泛用性不足.

3 负载选择加速方法

本节介绍负载选择加速方法的相关工作，它们基于微架构相关特征和(或)微架构无关特征衡量负载间相似性，只选择具有代表性的子负载集及其输入数据集进行后续评估，从而减少性能评估的时间.

负载选择方法期望所选的子负载集是有代表性的或非冗余的，即任何一对子负载集在不同的系统上达到的性能有明显的不同. 系统评估是有效执行的，即在被评估的系统上，有代表性的子负载集的平均性能接近于整个套件的平均性能. 同时，基准测试中一种常见的做法是使用较小的测试或训练输入数据集来减少模拟时间，通过选择较小的负载输入集或构造行为相似的数据集，减少动态指令数，从而缩短单次软硬件评估的时间.

表 3 对比了负载选择方法的工作，包含使用特征的方式、聚类算法、负载选择的数量和总负载数量的比值(即负载选择比)、平均 IPC 误差百分比，通过负载选择比可近似评估加速比. 微架构相关特征直接反映执行时间和功耗等性能指标，基于其的方法优势在于准确性较高，但缺点是结果会受到所选微架构配置的特性的影响，迁移性较差，且需要进行一定次数的评估以获取微架构相关特征，较为耗时，研究较少. 微架构无关特征可称为负载特征，刻画负载本身的行为，不受微架构配置的影响，因此可迁移性较强，同时仅需要如功能模拟的剖析工具即可快速获得，分析速度快，但缺点是准确性比仅使用微架构相关特征的工作低. 同时使用微架构相关和无关特征的方法综合了 2 种方法，因此在可迁移性和准确性上均表现优异. 基准套件是特定场景的负载集，用于评估处理器性能和功耗等，表 4 汇总了常用的基准套件及其简称.

3.1 基于微架构相关特征的方法

这类工作只使用与微架构相关的特征进行负载选择. 表 5 汇总了常用的微架构相关特征，如执行时间和 CPI. 文献 [15] 基于 PB(Plackett and Burman)设计，计算微架构参数对性能的显著性排名，并基于排名向量的欧氏距离表征负载相似程度，根据用户定义的阈值进行聚类. 文献 [27] 利用负载在不同多核处理器架构下的执行时间向量作为负载签名，以欧氏距离度量相似性进行层次聚类，其通过 SPEC OMP2006 公开数据进行了验证，选择 55% 负载的误差小于 5%.

3.2 基于微架构无关特征的方法

这类工作只使用微架构无关特征进行负载选择. 表 6 汇总了常用的微架构无关特征，包括指令混合、指令级并行性等. 微架构无关特征可用来刻画负载本身的行为. 其中，局部性的重用距离特征常用堆栈距离概要[59,130] 来表征. 这类研究可分为输入数据集选择和负载选择 2 类.

1)输入数据集选择

这类工作通过选择较小的负载输入集或构造行

Table 3 Comparison of Workload Selecting Methods

表 3 负载选择方法的对比

方法类型	方法来源	使用微架构相关特征的方式	使用微架构无关特征的方式	聚类算法	负载选择比	误差/%
基于微架构相关特征的方法	文献 [15]	参数显著性排名	✘	阈值聚类	7/12	-
	文献 [27]	执行时间向量	✘	层次聚类	6/11	5
基于微架构无关特征的方法	文献 [28]	✘	卡方检验	✘	-/23	-
	文献 [29]	✘	主成分分析	层次聚类	7/79	-
	文献 [30]	✘	基本块向量	距离最大	60/20 000	-
	文献 [31]	✘	主成分分析	k 均值聚类	9/21	15
	文献 [32]	✘	基本块向量+主成分分析	层次聚类	4/47	-
	文献 [33]	✘	分组主成分分析	共识聚类	-	-
	文献 [34]	✘	独立成分分析	多种聚类	5/27	3
	文献 [120]	✘	主成分分析/遗传算法	k 质心聚类	50/118	5
	文献 [121−122]	✘	凸壳体积、主成分分析	遗传算法	6/22	-
基于微架构相关与无关特征的方法	文献 [29,35]	主成分分析		层次聚类	14/29	-
	文献 [36]	主成分分析		层次聚类	10/23	-
	文献 [37]	主成分分析		层次聚类	12/43	7
	文献 [123]	多元因素分析		层次聚类	10/23	-
	文献 [38]	主成分分析+线性判别		多种聚类	20/54	-

注：“负载选择比”列中的“/”表示选择的负载数量和全部负载数量之比，“-”表示文献中无数据. “✘”表示无该项.

Table 4 Summary of Common Benchmark Suits

表 4 常用基准套件汇总

类型	工作负载	简称
多媒体和通信	MediaBench[124]	MediaBench
嵌入式	MiBench[125]	MiBench
单线程	SPEC CPU 2000[126]	SPEC2k
单线程	SPEC CPU 2006[127]	SPEC2k6
单/多线程	SPEC CPU 2017[18]	SPEC2k17
多线程	Princeton Application Repository for Shared-Memory Computers[128]	PARSEC
多线程	Stanford Parallel Applications for Shared Memory[129]	SPLASH

Table 5 Microarchitecture-Dependent Features

表 5 微架构相关特征

类型	特征
整体聚合	执行时间、CPI、功率
控制流	分支预测 MPKI、BTB 命中率
cache 行为（Icache/Dcache/L2/L3）	访问数量、命中数量、MPKI
TLB 行为（ITLB/DTLB/L2TLB）	访问数量、命中数量、MPKI

注：MPKI 表示每千条指令缺失.

Table 6 Microarchitecture-Independent Features

表 6 微架构无关特征

类型	子类型	特征
指令流	指令混合	整型、浮点、SIMD 等
		控制分支
		存储读/写
	寄存器通信	平均操作数数量
		平均使用次数
		重用距离
	指令级并行性	不同窗口大小的并行度
		基本块大小
	指令局部性	指令工作集大小
		时间、空间重用距离
数据流	数据局部性	数据工作集大小
		时间、空间重用距离
	通信特征	私有数据读写次数
		生产者写/消费者读次数

注：SIMD 表示单指令多数据流.

为相似的数据集，减少动态指令数，从而缩短单次软硬件评估的时间.

MinneSPEC[28] 是合理地模仿 SPEC2k 完整参考数据集的行为的小数据集，基于函数级执行模式、指令组合和内存行为，采用卡方检验测试选取的不同指令长度的数据集是否有显著性差异，在总共 33 个参考数据集负载中有 21 个非常接近，加速效果可达 2~4 个数量级. 该数据集已在文献 [90, 92] 中得到应用. 文献 [131] 验证了 SPEC2kInt 测试输入集不适合用于模拟，因为它们没有类似于参考输入运行的执行模式. 训练数据集比测试数据集在保持与参考输入集

类似的执行行为方面更好，但小输入集导致缓存缺失的执行路径非常不同. 文献 [29] 可靠地量化负载特征的相似性，利用主成分分析与层次聚类方法（PCA-H），分析 SPECint95 减少的输入集可导致与参考输入类似的负载行为. 文献 [30] 综合对比了 MiBench 的多个输入数据集，包括随机选择、基于微架构无关特征的选择（最大化基于基本块的曼哈顿距离）、过滤选择、最小-中间-最大选择的 4 种选择算法，发现可以用最少 3 个数据集找到最佳设计点.

2）负载选择

这类负载选择的方法对微架构无关特征进行降维，随后通过聚类算法选择具备代表性的负载集.

若干文献利用主成分分析方法进行特征降维. Joshi 等人[31] 利用主成分分析对特征维度进行降维，同时采用 k 均值和层次聚类算法选择代表性负载，并对 SPEC2k, MediaBench, MiBench 进行了验证，对 SPEC2k, 从 21 个负载中选择了 9 个，IPC 误差为 15%. Joshi 等人[132] 进一步利用相关性分析和遗传算法，加速表征过程，得到需要测量的特征更少，特征维度具有更直观的意义. 阶段复杂性曲面[32] 基于基本块向量刻画指令区间，利用阈值聚类将指令区间分类为阶段，利用主成分分析降维阶段特征，并采用层次聚类生成代表性的负载-输入子集. Eeckhout 等人[133] 评估了基于主成分分析与 PB 设计的统计负载选择方法，对比其他选择方法，并发现统计负载选择方法的绝对精度和相对精度较高.

基于主成分分析的变形方法，BenchSubset[33] 利用分组主成分分析，配合共识聚类，同时考虑了基准套件的通用性和多样性. CASH[34] 针对多线程负载增加 4 个并行的特征，利用独立成分分析进行降维，根据多个聚类指标的多数投票方法决定最优聚类方法，评估的聚类算法包含 k 均值、层次聚类、分裂层次方法、k 质心、装袋和共识聚类.

利用遗传算法同样可以达成降维和聚类效果. 文献 [120] 对比了主成分分析和遗传算法在降维方面的效果，以负载对之间的欧氏距离的相关系数作为评估函数，采用 k 质心聚类，在文献 [31] 的基础上增加了 BioInfoMark 等测试基准，发现遗传算法相比主成分分析方法提升了效果，选择 118 个负载中的 50 个，最大 CPI 误差仅 5%. 文献 [121] 根据微架构无关特征和负载覆盖阈值，考虑质心偏移、执行时间百分比、凸壳百分比的组合目标为适应度函数，采用遗传算法生成子集. 用质心来近似刻画负载空间，负载子集和原始套件之间的负载空间覆盖的差异、代表性（即由基准的凸壳体积表示的工作负载空间覆盖率）和效率（即总运行时间）为遗传算法的共同优化目标. 文献 [122] 进一步在 SubsetTrio 中采用主成分分析降维负载特征，整合遗传算法、3 维几何渲染方法提升效果.

3.3 基于微架构相关与无关特征的方法

这类工作同时使用微架构相关和无关特征进行负载选择，同样采用降维和聚类算法的结合.

利用 PCA-H 方法，Eeckhout 等人[29] 对 SPEC2k 的测试用例集的相似度和冗余度进行推断. Phansalkar 等人[35] 进一步对 SPEC2k6 进行类似的分析. 文献 [36] 对 SPEC2k17 采用相同的 PCA-H 方法，从 23 个 speed 负载中选择 10 个负载，加速 1.6 倍. 文献 [37] 进一步对 SPEC2k17 进行了细节分析，从 43 个负载中选择 12 个，达成 5.2 倍的加速，准确率达 93%，且与数据库、图分析和电子设计自动化等工作负载进行了性能特征的对比.

除了 PCA-H 方法外，文献 [123] 基于多元因素分析和层次聚类分析了 ImplantBench 套件. BenchPrime[38] 对特征进行主成分分析降维，使用线性判别分析生成签名，以评估贝叶斯信息准则的方式选择最佳聚类算法和最佳聚类数，根据负载集的多样性对基准套件进行排名，并评估每个基准套件与其他套件相比的独特性.

4 部分模拟加速方法

本节对比部分模拟加速方法，通过选取部分指令用于后续评估从而加速负载评估的过程，包括统计采样模拟和综合模拟，如表 7 所示.

统计采样模拟只对原始负载的部分指令序列进行模拟，关注点在于考虑绝对精度或相对精度下优化估计的性能指标与全指令的性能指标的偏差. 综合模拟方法并不采样原始指令序列，而是基于微架构无关的指令流和数据流等特征，生成很短的克隆负载，使得克隆负载的性能指标与源负载的性能指标接近.

统计采样模拟的优势在于采样难度较低且偏差较小，缺点在于加速效果较小，且需要额外的存储开销. 综合模拟的优势在于加速效果较好，可灵活调整所需行为模式，但模拟精度较低，且构建难度较高.

4.1 统计采样模拟

统计采样模拟通过只模拟部分指令片段来估计整个负载的性能指标. 我们将统计采样模拟分为 3 类，

Table 7 Comparison of Partial Simulation Acceleration Methods

表 7 部分模拟加速方法的对比

类型	目标	子类型	方法来源	指令流	数据流	微架构相关特征	加速比	误差/%
统计采样模拟	采样单线程	随机采样	文献 [134]	✘	✘	✘	-	7~17
		均匀采样	文献 [39]	✘	✘	✘	35~60	0.6
			文献 [40]	✘	✘	✘	~4 000	3.5
		代表性采样	文献 [41−42,135−136]	✔	✘	✘	62~107	3.7
			文献 [43]	✔	✘	✘	1~1.4	0.5
			文献 [137]	✔	✘	IPC, cache	~100	3
			文献 [138]	✔	✘	IPC, cache	-	2~8
	采样多线程	基于时间	文献 [44]	✔	✘	IPC	10	5
			文献 [139]	✔	✘	IPC	5.8	3.5
			文献 [45]	✔	✘	IPC	20	5.3
		基于负载和特定同步	文献 [47]	✔	✘	✘	25	0.9
			文献 [48]	✔	✘	✘	220	0.5
		基于循环迭代	文献 [46]	✔	✘	✘	801	2.3
	采样访存	基于检查点	文献 [49]	✘	✔	cache, BP	8 000~15 000	~0.6
			文献 [51]	✘	✔	cache, BP	50~100	~0.6
			文献 [50,140−141]	✘	✔	cache, BP	-	-
		基于预热	文献 [142]	✔	✔	cache, BP	8 000~15 000	~0.6
			文献 [143]	✔	✔	cache, BP	~100	1.5
			文献 [144]	✔	✔	cache, BP	~70	0.3
			文献 [145−147]	✔	✔	cache, BP	-	-
综合模拟	综合单线程		文献 [54]	✔	✘	cache, BP	-	5~7
			文献 [148]	✔	✘	cache, BP	-	4.1
			文献 [149]	✔	✘	cache, BP	-	-
			文献 [52−53]	✔	✘	cache, BP	-	8
			文献 [150]	✔	✘	cache, BP	~1 000	6.6
			文献 [55,151]	✔	✔	cache, BP	~1 000	2.4
			文献 [116]	✔	✔	cache, BP	520	5.1
			文献 [152]	✔	✔	cache, BP	-	3.2
			文献 [153−155]	✔	✔	✘	-	-
	综合多线程		文献 [58]	✔	✔	✘	9~385	3.8~9.8
			文献 [156]	✔	✔	✘	1 000~10 000	4.9
			文献 [56−57]	✔	✔	cache, BP	40~70	5.5
			文献 [157]	✔	✔	cache, BP	21	8
	综合访存		文献 [59]	✘	✔	✘	-	0.4~3.1
			文献 [60−61]	✘	✔	✘	-	-
			文献 [158−160]	✔	✔	✘	31	4.8
			文献 [161]	✔	✔	✘	20	2.8
			文献 [62]	✔	✔	✘	20~50	4.2
			文献 [162]	✔	✔	✘	-	9

注: “-”表示文献无该数据. “✔”表示有使用该类数据, “✘”表示没有使用该类数据.

包括采样单线程、采样多线程、采样访存, 分别关注单线程采样代表性、多处理器同步、存储开销方面.

1)采样单线程

采样单线程的统计采样方法可分为随机采样方

法、均匀采样方法、代表性采样方法.

①随机采样方法随机选择起始的点和指令片段长度. 文献 [134] 关注于分支目标缓存的状态，对采样配置和采样偏差进行了分析，建议置信度低时通过增加采样数来提高.

②均匀采样方法不选择特定指令片段，依靠统计学的大数定律使得估计结果接近真实结果，也称周期性采样. 如图 2 所示，统计采样模拟（SMARTS）[39] 采用若干相等长度的指令片段以及相等指令间距，使用统计抽样理论来估计采样模拟相对于参考模拟的 CPI 误差，在 CPI 误差平均 0.64% 的情况下加速 35 倍. 如果估计的误差高于用户指定的置信区间，那么建议采用更高的采样频率. LiveSim[40] 创建负载架构状态的内存检查点，对若干检查点进行增量式并行模拟，并报告置信区间，采样越多，结果越准确，并能实时更新.

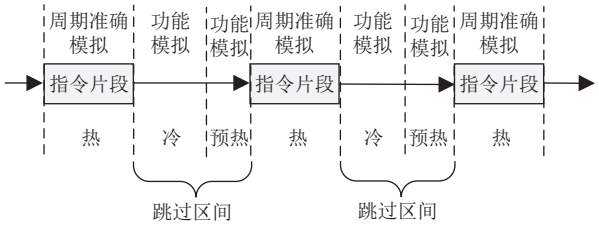

Fig. 2 Statistical sample simulation

图 2 统计采样模拟

③代表性采样方法选择特定的样本. SimPoint[41-42] 是一种广泛使用的指令采样技术，其将负载划分为定长的指令片段，基于基本块刻画指令片段行为，以曼哈顿距离计算基本块向量之间的相似度，利用 k 均值聚类，得到加权的指令片段集. PinPoints[135] 使用 Pin 工具动态插桩来辅助 SimPoint. 文献 [136] 为 SPEC-2k6 生成 SimPoint 数据. LPP[43] 基于堆栈重用距离，利用可变距离采样、小波滤波和最佳片段划分来识别局部片段，进而通过语法压缩构建指令片段层次结构. 文献 [137] 基于 CPI 和 cache 缺失分析指令片段之间的相似性，构建了一个组合的分层过程调用和循环图来表示负载的执行. VLI[138] 在负载划分执行阶段，基于微架构相关特征，对每个阶段通过主成分分析和 C5.0 聚类，生成加权的阶段集合作为代理负载. Co-Phase Matrix[163] 研究使用单独的负载阶段信息来指导 SMT 模拟，通过收集负载阶段组合的样本来填充共阶段矩阵，并用于指导模拟点之间的快速前进.

文献 [164] 综合对比 SimPoint 和 SMARTS，发现两者在速度和精度之间取得了最好的权衡，并提出了一种决策树，从精度、速度-精度权衡、配置依赖分析 3 个维度帮助选择最合适的模拟技术. 文献 [19] 对比随机采样和均匀采样，如果一个负载具有周期性的行为且采样频率接近基准的自然频率，那么均匀采样就会比随机采样精度更低，即使随机采样需要更少的测量.

2）采样多线程

多线程负载天生就难以分析，因为线程可以在任何时候进入睡眠状态，并且线程之间会相互干扰，常规的负载参数会产生复杂的行为，比如线程与核心的不对齐和不平等的 cache 分布. 因此如何对多线程负载进行采样存在很大挑战. 采样多线程的方法可分为 3 类，分别是基于时间的采样方法（time-based sampling, TBS）、基于负载和特定同步的采样方法、基于循环迭代的采样方法.

①基于时间的采样方法. ESESC[44] 首先在多核处理器模拟中支持采样，根据前进时间而不是指令数对执行进行采样，几乎不受应用负载类型（多进程或多线程）、核数量、模拟配置的同质性或异质性的限制，平均 5% 的 IPC 误差下可达约 10 倍加速. MTASM[139] 保持在模拟快速前进期间的线程交互，提出预处理步骤来确定有效的采样参数，以识别周期性，并确定保持良好一致性的采样机制，以避免混淆问题. 基于分形特性的采样方案 PCantorSim[45]，发现 IPC 变化随着时间的变化表现出分形或自相似行为.

②基于负载和特定同步的采样方法. BarrierPoint[47] 将屏障间的区域作为待采样区间，平均 0.9% 的执行时间误差可达 25 倍加速. 文献 [165] 对 BarrierPoint 引入了可变的屏障点粒度，支持基于不同屏障点类型的屏障点微调. TaskPoint[48] 使用任务（指定的指令片段）作为待采样区间，利用 DBSCAN 聚类基本块，采用机械模型平衡模拟速度和准确性，使得采样复杂性受限于负载多样性，而不是负载长度，平均 0.5% 的执行误差下可达 220 倍加速.

③基于循环迭代的采样方法. LoopPoint[46] 扩展文献 [137] 中面向单线程的以循环迭代作为工作单元的方法，该方法既不限制所使用的同步原词的类型，又能根据负载显示的相似性进行伸缩，利用负载知识或同步机制的细节以避免同步循环项作为区域边界，平均 2.33% 的执行时间误差下可达 801 倍加速.

3）采样访存

采样访存的工作通过检查点机制和预热机制对架构和微架构状态进行恢复，加速采样模拟. 检查点机制通过存储架构状态信息，可以显著减少采样点状态恢复的时间，但需要额外的存储开销. 预热机制

通过采样点模拟前预先快速运行一段指令或设置微架构状态的形式, 解决冷启动使得微架构状态不精确导致的性能估计偏差大的问题. 这 2 种机制常常配合使用.

①基于检查点的采样模拟加速. 检查点机制无需重复执行冗余或影响微架构有限的前序指令, 通过存储架构、微架构的执行状态, 再次执行时直接恢复状态信息, 极大减少了非必要的模拟时间开销. SimFlex[49] 为单处理器负载设计 live point 检查点以恢复以微架构状态, 为多处理器负载设计 flex point 检查点以额外恢复目录标签和完整的存储数据. 反向状态重构[51] 方法在模拟点之间跳跃时, 记录重构所需的数据, 模拟时反向扫描这些数据, 近似化地恢复状态. 香山[50] 提出了一种 RISC-V 格式的检查点机制, 通过若干系统级指令配合存储数据生成检查点. PinPlay[140] 基于 Pin 创建允许重复分析的用户级检查点, 其包含一个内存映像(所有线程通用的指令)和区域开头的架构寄存器值, 后者包含每个线程的一个寄存器文件、来自系统调用和信号的寄存器修改信息. ELFies[141] 基于 PinPlay 创建独立可执行文件.

②基于预热的采样模拟加速. 预热机制通过采样点模拟前预先快速运行一段指令或设置微架构状态的形式, 解决冷启动使得微架构状态不精确导致的性能估计偏差大的问题.

使用功能预热, SMARTS[39] 维护了分支预测器和缓存状态, 并在采样点间快速前进. TurboSMARTS[142] 将功能预热状态存储在小型、可重用的检查点库中, 从而消除功能预热瓶颈. 针对采样的易用性, 透明采样[143] 根据模拟代码片段的需要自适应地进行预热, 平衡采样技术的易用性和效率.

基于内存引用的方法中, 内存引用重用延迟(MRRL)[145] 是指每个唯一内存位置的连续引用之间完成的指令数量, 在每个模拟点之前选择一个点来进行缓存层次结构和分支预测器建模, 就可以快速准确地建立状态, 平衡采样模拟的高精度和快速性的要求. 边界线重用延迟(BLRL)[144] 将重用延迟(对同一内存位置的内存引用)跨越模拟点前和模拟点之间的边界, 以计算每个模拟点所需的预热, 比 MRRL 加速 1.5 倍. 进一步, NSL–BLRL[146] 结合无状态损失(non-state-loss, NSL)和 BLRL 技术, 使用模拟点前最近最少使用的内存引用流对 cache 预热, 检查点模型下比 BLRL 加速 14.0 倍. SOL[147] 针对基于时间的多线程采样, 采用初始预热和扩展的无状态损失实现在线预热, 解决存储开销过大和多次运行时性能指标不稳定的问题.

4.2 综合模拟

综合模拟(synthetic simulation), 也称统计模拟(statistical simulation)、负载克隆(workload clone). 图 3 为综合模拟的典型框架流程. 首先对负载采用微架构无关剖析工具和局部性事件的特化模拟获取统计概要(profile), 收集微架构相关和无关特征. 其次, 基于统计概要文件生成综合踪迹(trace). 最后对综合踪迹进行模拟以获取性能和功率等指标. 这种综合模拟框架背后的关键思想是识别影响真实负载性能和功耗的关键特征, 如指令混合、线程级并行、内存访问行为、分支可预测性等, 并通过控制这些特征的值来生成综合的可执行负载. 模拟的指令序列长度比源负载短很多, 且可只关注微架构组件资源调度如指令窗口、功能单元和重定向缓冲器, 无需计算真实的数值, 因而模拟速度比周期准确的模拟器快. 此外, 综合模拟克隆原始工作负载的行为, 无需提供源码, 缓解了软件的专有性或保密性的问题. 综合模拟按照工作的重点可分为 3 类, 分别是综合单线程、综合多线程、综合访存.

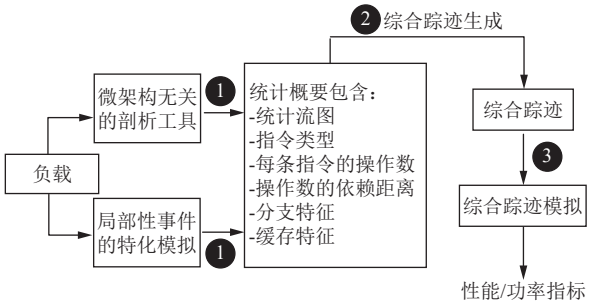

Fig. 3 Synthetic simulation flow (SFG[69])

图 3 综合模拟的流程(SFG[69])

1)综合单线程

综合单线程的工作通过对单线程负载进行行为刻画, 生成与该负载有类似微架构行为的综合踪迹.

仅利用指令流的微架构无关特征, 文献 [54] 提出统计模拟器 HLS, 将应用负载的统计配置文件作为输入, 基于基本块信息来配置控制流信息, 从该配置文件动态生成代码库, 并在一个超标量微处理器核心的功能模式上执行此统计代码, 性能平均误差为 5%~7%.

Bell 等人[148] 基于 HLS[54] 提出 HLS++, 对微架构模型进行修正, 对每一对独特的前继和后继基本块进行测量, 而不是仅仅针对独特的单一基本块, 耗费 HLS 2 倍的时间, 将性能误差从 15.5% 降低到 4.1%. 文献 [149] 进而提出 SS-HLS++, 讨论了综合模拟的 3

个方面，包含性能瓶颈的保真度、跟踪设计更改的能力，以及准确性和统计模拟模型的复杂性，配置间加速比的相对精度超过 95%.

Nussbaum 等人[52] 提出统计模拟（statistical simulation, SS），开发了多个不同详细程度的综合踪迹生成模型，包括 cache、分支预测、指令依赖的信息，分析不同混合程度的模型精度，包括是否考虑基本块分布、分支距离、块依赖等，对误差和收敛性进行了详细的研究，性能误差为 8%. 文献 [53] 进而分析不同微架构参数下的精度，增加能耗延迟积的相关验证. SFG[150] 提出使用统计流图来描述负载执行的控制流，考虑分支预测的延迟更新，利用先进先出缓冲模拟了分支预测器的查询和更新，平均性能误差是 6.6%，平均能耗误差是 4%.

在指令流的基础上再利用数据流特征方面，Bell 等人[55,151] 提出的自动测试综合（automatic testcase synthesis, ATS）考虑统计流图和存储访问模型，生成汇编级代码组成踪迹，并在之后[166] 对能耗进行了实验验证，加速效果可达 3 个数量级，IPC 误差控制在 2.4% 以内. PerfProx[116] 对每个基本块的大小、依赖距离、内存访问流、系统调用和分支逐步生成代理，原始数据库应用为生成的代理基准的 520 倍，而平均 IPC 误差仅 5.1%. Datamime[152] 通过改变负载输入数据集，基于贝叶斯优化迭代选择数据集参数使得性能指标的分布距离最小. 除了统计流图外，Joshi 等人[153] 提出方法 Performance Cloning，只利用微架构无关特征进行分析，并提出使用遗传算法自动生成功率压力基准的框架 StressMaker[154-155].

2）综合多线程

针对多线程负载及多核处理器设计，综合多线程的工作考虑负载多线程之间的同步和资源共享竞争的问题. 相比单核处理器模拟器，芯片多处理器模拟器将会使仿真速度减慢[167]，因此有必要针对多处理器的环境进行加速.

①基于微架构无关特征. SSFG[58] 基于 SFG[150] 提出的统计流图，构建同步统计流图，其中包含线程间同步和共享行为，以捕获多线程的复杂特征和交互信息，并开发了线程感知的数据引用模型和基于小波的分支模型，以生成准确的内存访问和微架构无关动态分支统计信息，IPC 平均误差为 3.8%~9.8%，可以减少一个量级的模拟时间. Ganesan 等人[156] 基于之前 MLP 感知的工作[161]，并借鉴文献 [151] 的生成算法，提升内存访问、分支和指令并行性模型，考虑的组件包括共享缓存、一致性逻辑、乱序核、互连网络和 DRAM 等，IPC 平均误差为 4.87%，功耗平均误差为 2.7%，达成 4~5 个数量级的加速.

②增加微架构相关特征. 文献 [56–57] 基于 SFG[150]，增加对共享资源（如共享缓存和片外带宽）建模，以独立于微架构的方式建模内存地址流，收集缓存集访问概率和堆栈距离概要来建模，并通过对每个指令片段生成合成踪迹以建模负载的时变执行行为，IPC 平均误差 5.5%，达成 40~70 倍加速. 文献 [57] 进一步放宽固定延迟的 DRAM 模型的假设，使用内存地址流扩大了内存层次的探索空间. MINIME[157] 利用并行模型捕获多线程应用的特征，包括任务、数据和数据流 3 种类型，所生成的可移植的基准，既保留了高级的特性，也保留了低级的特性，平均性能误差为 8%，加速 21 倍.

3）综合访存

综合访存的工作关注缓存和存储器层次，通过对内存访问模式进行建模，以此生成综合踪迹.

仅利用数据流微架构无关特征，WEST[59] 利用组栈距离（set stack distance, SSD），对 cache 的每个组限制一个私有值集，捕获数据访问流中的时间局部性. STM[60] 在 WEST 的基础上，使用步长历史模式索引的转移概率表来建模多个具有不同步幅值和不同步幅长度的流，增加了克隆空间局部性的能力.

同时利用指令流和数据流的微架构无关特征，Lee 等人[158] 评估了 3 种模型来量化长延迟内存访问的影响，包括隔离缓存未命中模型、独立缓存未命中模型以及成对依赖缓存未命中模型（PDCM）. 文献 [159] 进而提出 In-N-Out，在踪迹生成期间，仅使用功能缓存模拟器而非周期级模拟器，利用指令之间的数据依赖关系信息来估计 L1cache 缺失之间的距离，并对非核组件如缺失状态处理寄存器和数据预取部件进行建模. 文献 [160] 进而优化 PDCM，动态重建重排序缓冲区状态并遵循踪迹项之间的依赖关系，确定如何处理与踪迹中记录的其他缓存缺失相关的缓存缺失. 文献 [161] 利用 MLP 感知的形式生成踪迹，以估计对主存访问的突发性. MeToo[62] 增加了将内存请求的时序行为克隆到 STM 的能力，设计了指令计数间隔的方法来模拟存储反馈循环. SLAB[162] 依靠指令流局部性和全局地址流模式降低了元数据的存储开销.

近年来，基于综合模拟新提出了克隆变形（clone morphing）的工作，目标是产生新的假设工作负载，这些工作负载具有当前不存在的性能行为. Clone Morphing[61] 扩展了 STM，以层次化地址历史表代替步长历史表，从时间局部性、空间局部性和内存占

用3个维度进行分析,既可以用于预测未来的工作负载,也可以用于获得2个不同现有工作负载的平均行为,从而填充行为映射.

5 设计点选择加速方法

本节总结设计点选择加速方法,目标是选择部分的设计点,使其能代表整体设计空间,探索得到最优的目标设计.设计点选择方法分为2种,采样方法和迭代搜索方法.采样方法通过一次性选取设计空间中的若干设计点,而迭代搜索方法通过逐轮确定待探索的设计点.二者常配合使用,先使用采样方法生成初始采样集,随后应用迭代搜索方法.此外,设计点选择加速方法常与第7节的预测模型结合使用,文献[14, 25–26, 67–68, 75, 97, 168]同时涵盖设计点选择与预测模型加速,本节将重点放在设计点选择加速方法的对比上.

5.1 采样方法

采样方法通过一次性选取设计空间中的若干设计点,只模拟该设计点集加速探索过程,常用于初始化采样集.采样的设计点简称为采样点.采样方法包括基于参数敏感度和基于实验设计的2类方法.

1)基于参数敏感度的采样方法

本节总结基于设计空间参数的敏感度的采样方法,通过选择对性能指标具备高敏感度的参数,减少所需要探索的设计空间,加速探索过程.

文献[63]对功能单元设计空间,基于功能单元的使用率,以加权求和作为敏感度,敏感度高表示该功能单元对架构的影响大,探索中应给予重视,因而选择敏感度最高的30%的功能单元变量进行全空间探索,而其余变量启发地选择值.文献[64]探索cache微架构能耗延迟积(energy delay product, EDP)的权衡,将参数相对于全搜索最优配置的敏感度定义为当其变化可能的最小量时,度量相对于最优情况的最大变化.文献[65]以能耗延迟积作为代价函数,计算代价函数对于参数和负载的敏感度,再进行局部搜索优化微架构参数.文献[66]面向嵌入式处理器,定义参数敏感度为加速比与处理器面积的比值,进行类似搜索过程.

文献[15]为每个参数分别选择低值和高值构建显著性矩阵,对参数的重要性进行排序,使用统计方差分析(analysis of variance, ANOVA)技术对每个关键参数进行迭代的敏感性分析.根据敏感性分析的结果,选择关键参数的最终值,其通过比较应用增强方法前后各参数的排名之和,分析参数对处理器性能的提升效果.排名累加值仅代表重要程度,无法表示量级.同样利用ANOVA,文献[67]计算每个参数的贡献百分比,允许分离单个参数来评价对性能指标的影响,根据最新的实现来选择参数及其值的范围.文献[169]采用树模型,在树模型中的每个节点选择一个变量对它的范围进行分裂,而变量敏感度是基于其被选择分裂的次数,分裂次数越多,代表该变量对预测结果的影响越大.

2)基于实验设计的采样方法

本节总结基于实验设计(design of experiment, DoE)的采样方法.良好的采样方法应该对设计空间均匀采样,同时保证采样点具有代表性.

表8汇总了不同的基于实验设计的采样方法.

Table 8 Comparison of Design of Experiments

表8 实验设计的对比

类型	实验设计	样本数
随机	均匀随机[3,170]	M
基于参数级别	2级全阶乘[25]	2^N
	中心复合设计[25]	$1+2N+2^N$
	Box-Behnken[25]	$1+2^N$
	PB设计[15]	$2N$
	正交设计[6-7]	相对固定
	拉丁超立方体[92,97]、均匀拉丁超立方体[34,171]	M
基于距离	智能采样[3]	M
	最小的成对距离最大[169]	M
	最大化距离矩阵的迹[14]	M
	k均值[172-173]	N

注:M为所需的样本数,N为参数个数.

①随机方法.均匀随机是最常见的一种采样方法[3,170],基于统计学理论,很好地覆盖全设计空间.

②基于参数级别的方法.参数可选值的数量对应参数的级别(level),参数也称为因素.这类方法试图在设计空间上选取有代表性的设计点.

文献[25]对比了若干种采样方法.全阶乘(full factorial)设计,也称全阶乘设计,是指所有参数最大值和最小值的全组合,可以评价每个参数之间相互作用的影响.中心复合设计是一种专门针对2次响应面结构的实验设计,设计包括以下3套不同的采样点:第1套为2级全阶乘或分数阶乘设计,2级全阶乘即每个参数有2种可能取值,采样点为所有组合,分数阶乘设计为从中抽取一定比例,如1/2或1/4;第2套为一组中心点,即每个参数是在阶乘部分中使用

的值的中位数；第 3 套为一组轴向点，即除了 1 个参数，其余参数与中心点相同，这个参数将在低于和高于 2 级的中位数上取值（如参数范围的上下边界）. Box-Behnken 设计适合于参数组合在空间边的中心和所有参数在中心的 2 次模型，主要的优点是能够避免参数组合在同一时间采用极端值（与中心复合设计相反）. PB 设计[15,149]是一种常用的实验设计方法，要求任何一对参数的级别组合都出现相同次数. 然而，每个参数只考虑 2 个层次，不适合多级别的多参数的设计空间.

正交设计根据正交数组（orthogonal array, OA）选择少量的代表点，均匀地覆盖整个设计空间，被 ActBoost[67] 等采用. 每个参数的每个级别出现相同的次数，任意 2 个参数的每个可能的级别组合出现相同的次数. 正交设计的效果较优. 然而，为了实现数组的正交性和均匀性，正交设计的样本大小相对固定. 随着参数数量和参数级别的增加，很难找到相应的正交数组，不能灵活地选择任意数量的样本来进行评估. TrEE[6] 进一步提出基于正交方法的折叠设计，翻转或者转换若干参数的级别，可灵活扩展采样集大小.

拉丁超立方体采样（Latin hypercube sampling, LHS）[92,97] 面对N个参数，每个参数被均匀分成M个区间，M为所需的采样集大小. 从中随机选择M点，以确保每一个参数的每个级别被研究 1 次. CASH[34] 利用均匀拉丁超立方体，先基于 PB 方法[15] 构建树，前序遍历到树的m级，$m+1$级均匀随机，最终生成M个设计点.

③基于距离的方法. 这类方法基于选定的距离函数评估设计点之间的相似性，以此选择相距较远的设计点以更好地代表全空间. 文献 [3] 提出智能采样，根据多个神经网络模型预测的方差系数排序，按序对超过距离阈值的设计点进行采样. 文献 [169] 最大化最小的成对距离，距离按参数级别加权. 文献 [14] 利用微架构感知采样 MicroAL，通过迭代最大化在新采样设计和未采样设计上构造的距离矩阵的迹，可以获得具有高互发散的代表性特征向量池. McPAT-Calib[172-173] 对设计点进行 k 均值聚类获取初始集采样集.

5.2 迭代搜索方法

本节总结迭代搜索方法. 迭代搜索方法通常依据目前的搜索结果，通过建立代理模型估计未采样点的特征，依据获取函数确定下一轮采样的设计点，设计点通过评估后重复该过程，直到满足用户需求. 迭代搜索方法可分为 3 类，分别是启发式、组合优化、统计推理方法.

早期的工作中，启发式方法多通过贪心策略，算法简单且耗时少，但效果一般. 遗传算法作为组合优化方法中的一种被多个工作采用，与帕累托目标结合算法 NSGA-II 的效果被证明优于一般的启发式方法. 近年来，统计推理的方法进一步提升了探索效果，常配合贝叶斯优化算法，基于方差系数、预期改善、预期超体积改善等获取函数平衡勘探和利用.

我们将迭代搜索方法按照 3 个维度进行对比，包括代理模型、搜索/获取函数、设计空间，如表 9 所示. 代理模型表示获取性能指标的方法，部分方法[72,74] 通过 2 层次模拟，组合高层次低精度（如事务级模拟）和低层次高精度（如周期级模拟）的模拟方法加速评估过程. 部分方法[74-75,180] 预测帕累托非支配级别而非性能指标以辅助搜索算法. 搜索/获取函数表示决定下一轮采样的设计点的策略.

硬件设计空间表示所探索的微架构参数范围，早期多数工作探索单核 CPU 微架构参数和 VLIW 处理器参数，随着多核处理器的兴起，核数和存储层次等更多的参数被纳入设计空间中.

1）启发式方法

启发式方法通常依靠参数敏感度等启发式信息进行贪心搜索或部分空间全搜索，期望搜索到的结果为全局最优.

Platune[174] 采用贪心策略，首先构建参数依赖模型，寻找强连通子图作为聚类，计算 1 个聚类中的帕累托最优解，之后将 1 对聚类组合成 1 个聚类，并在其中计算帕累托最优配置，迭代组合过程直到剩 1 个聚类. 文献 [64] 根据敏感度排序，基于贪心探索变量的最优取值，独立计算局部 Icache 和 Dcache 的最优配置，组合为全局最优配置. Sheldon 等人[66] 将设计空间简化为 0-1 背包问题，定义参数敏感度为加速比除以面积的值，基于负载特化或固定的参数敏感度排序，贪心地按影响从高到低的顺序搜索参数. 文献 [185] 扩展单因素分析，利用参数相互影响的 2 级全阶乘实验设计得到参数敏感度排序，再利用相同方法进行搜索. 文献 [175] 将二进制搜索树的空间修剪方法用于 VLIW 处理器探索，参数先在独立的空间优化，再选择其他参数进行探索. 文献 [176] 将参数分类为重要、次重要、不重要 3 类，分别采用穷举探索、贪心、初始集最优值的探索方法，再将设计分为偏重性能还是偏重功耗 2 类，以 2 个权重值将 2 目标问题转换为单目标问题.

Table 9 Comparison of Iterative Searching Acceleration Methods

表 9 迭代搜索加速方法的对比

类型	子类型	方法来源	代理模型	搜索/获取函数	硬件设计空间
启发式		文献 [174]	-	参数聚类、贪心	单核片上系统
		文献 [64]	-	敏感度、贪心	cache 微架构
		文献 [66]	-	敏感度、贪心	FPGA 软核
		文献 [175]	-	二进制搜索树	VLIW
		文献 [176]	-	贪心、单目标化	CMP
组合优化	遗传算法	文献 [71]	-	GA	单核片上系统
		文献 [72]	2 层次模拟	局部搜索+GA	单核 CPU
		文献 [73,177]	模糊系统	GA	VLIW
		文献 [171]	多项式回归	GA	单核 CPU
		文献 [117]	-	爬山/GA/蚁群	CMP
		文献 [74]	ANN 预测级别	NSGA-II	CMP
		文献 [69]	ANN	NSGA-II	CMP
		文献 [178]	ANN	NSGA-II	VLIW
		文献 [68]	ACOSSO	NSGA-II	CMP
	模拟退火	文献 [178]	ANN 预测级别	模拟退火	VLIW
		文献 [179]	多种模型之一[25]	多种搜索算法	CMP
统计推理	不确定度	文献 [67,97]	AdaBoost.ANN	CoV	单核 CPU
		文献 [172−173]	XGBoost	距离的最小值	单核 CPU
	预期改善	文献 [75,180]	克里金模型预测级别	EI(+GA)	CMP
		文献 [34]	随机深林	EI	CMP
	超体积改善	文献 [13]	ACOSSO	EHVI	CMP
		文献 [14]	高斯过程	EHVI	单核 CPU
		文献 [181]	AdaGBRT	HVI+均匀性	单核 CPU
		文献 [182]	BagGBRT	HVI+UCB	单核 CPU
	帕累托	文献 [25]	多种模型之一	候选帕累托最优解集	CMP
		文献 [183−184]	马尔可夫决策	帕累托覆盖	CMP
		文献 [26,168]	马尔可夫网预测分布	帕累托最优解集	CMP

注: "-"表示该方法只以软件模拟或基于 RTL 的电路评估的方式获取性能指标, 其余方法可通过训练代理模型替代软件模拟来获取指标或指标之间的关系.

2)组合优化方法

组合优化方法寻找离散变量的最优编排、分组、次序或筛选等. 遗传算法和模拟退火模仿生物和物理机制的方法, 常用于组合优化问题, 搜索效果优于早期的启发式方法.

遗传算法, 也称为进化算法, 通过模仿生物基因自然选择的策略优化个体.

文献 [71] 基于 Platune[174], 在子空间大于阈值时采用遗传算法代替全空间搜索. 文献 [72] 采用遗传局部搜索算法用于探索乱序处理器设计的单目标和多目标设计空间, 使用随机下降算法对每一代中的每个设计点进行充分优化, 即在满足每一代中每个设计点的约束条件的同时, 对目标函数进行优化, 实验对比了帕累托模拟退火、帕累托反应性禁忌搜索和随机搜索. MOEA+Fuzzy[73,177] 采用多目标进化算法(multi-objective evolutionary algorithms, MOEA)探索 VILW 处理器的设计空间, 并使用 Takagi–Sugeno 模糊系统来估计要优化的性能指标. GPRS[171] 中候选设计点的性能指标是用表达式树表示的多项式方程, 其中树中的单个节点表示用户定义的操作符、编码的设计参数或调优参数, 利用遗传算法中的操作对树节点进行变换. Magellan[117] 应用爬山法、遗传算法、蚁群算法搜索算法, 提出 2 种标记方法, 基于对核心敏感的标记负载和基于对微架构组件的压力标记核心, 通过只模拟特征匹配超过核数的组合, 剪枝了搜索空间.

非支配排序遗传算法(nondominated sorting genetic algorithm, NSGA)是在遗传算法中增加考虑帕累托最优解的算法. Mariani 等人[74] 提出 ML-NSGA-II, 借助人工神经网络预测帕累托非支配级别, 按照比例分配低层次周期级模拟和高层次事务级建模的数量, 基于 GA 进行探索. 文献 [69] 进而采用 NSGA-II 搜索帕累托最优解, 利用人工神经网络预测执行时间和能耗. 针对 VLIW 处理器, 文献 [186] 在文献 [178] 的基础上将搜索算法由 MOSA 替换为 NSGA-II 提升了探索效果. MA-NSGA-II[68] 采用 ACOSSO 作为代理模型, 辅助 NSGA- II 探索执行时间和功率的帕累托最优解集.

模拟退火模仿固体在某一恒定温度下趋于热平衡的过程. 文献 [187] 采用帕累托模拟退火, 通过计算与所选优点相关配置的近似帕累托集进行探索. 文献 [178] 利用多目标模拟退火(multi-objective simulated annealing, MOSA)方法, 探索 VLIW 处理器的帕累托最优解, 使用人工神经网络预测设计的质量, 然后用它来决定是否应该模拟设计. MULTICUBE[179] 是一个集成框架, 包含 NSGA-II 和 MOSA 在内的多种算法. 文献 [24] 提供了一个公平的比较, 根据算法进行分类, 使用大量的指标进行综合评估, 并考虑初始设置工作、收敛速度、可伸缩性和质量的优化.

3)统计推理方法

本节总结基于统计数据进行推理的设计点选择方法, 典型地表现为贝叶斯优化方法, 其利用代理模型(或称为响应面模型)对设计点进行预测, 对未采样的设计点基于获取函数评估特定值, 从而决定下一轮探索的方向. 我们按照获取函数, 将相关工作分为 4 类, 分别是基于不确定度、基于预期改善(expected improvement, EI)、基于超体积改善(hypervolume improvement, HVI)、基于帕累托的迭代搜索方法.

①基于不确定度. 利用方差系数(coefficient of variation, CoV)表示不确定度, Li 等人[67] 提出 ActBoost, 通过集成人工神经网络模型评估不同设计点的 CoV, 进行后续设计点的选择. 文献 [97] 后续提出 SemiBoost, 采用半监督学习的思想, 对具有高自信预测的未标记样本进行伪标记, 以扩大训练数据集, 进一步提高预测精度. 利用距离表示不确定度, McPAT-Calib[172] 利用 iGS 策略, 对已采样和未采样设计点的特征距离和指标距离的最小值最大化. 文献 [172] 后续提出 PowerGS 策略, 增加微架构特征和负载维度.

②基于预期改善. 与 CoV 关注平均预测情况不同, EI 通过计算最小值提升的期望来最终最小化特定的性能指标, Mariani 等人[180] 利用基于指数相关函数的克里金法模型预测帕累托非支配级别分布, 基于非支配级别的 EI 进行迭代优化. 文献 [75] 将文献 [180] 扩展为 OSCAR, 在每轮中新设计遗传算法以获取最大化 EI 的设计点. CASH[34] 对比若干机器学习模型, 实验中发现利用基于随机森林作为代理模型, EI 作为获取函数的贝叶斯方法达到较优的探索效果.

③基于超体积改善. HVI 是在多目标空间中评估超体积改善, 文献 [13] 利用 EHVI(expected HVI), 采用基于模型不确定性的一种自适应组件选择和平滑算子(ACOSSO)的代理模型. BOOM-Explorer[46] 同样采用 EHVI 指标, 面对单核处理器微架构, 利用高斯过程模型本身对方差的估计效果, 探索性能与功率这 2 个目标的帕累托最优解集. MoDSE[181] 在 HVI 的基础上, 考虑通过帕累托最优解的均匀性来跳出局部最优解. MoEnDSE[182] 通过多个 GBRT 模型的预测方差估计不确定度, 基于配合上界置信边界的 HVI 方法平衡勘探和利用.

④基于帕累托. 基于帕累托集的工作包括利用帕累托最优解集或非支配级别的性质. ReSPIR[25] 利用特定于应用负载的约束来确定最大数量的可行解决方案, 在候选帕累托最优解集中包含违反负载特定约束的最有前途的配置, 其被定义为违反尽可能少的约束和最小惩罚的解. Beltrame 等人[183] 采用离散马尔可夫决策过程(Markov decision process, MDP)在设计空间中行走, 改变其参数, 只有在概率信息不足以做出决定时才进行模拟, 并提出一个学习算法在进行模拟时更新决策结果的概率. 文献 [184] 进而将 MDP 扩展为多目标 MDP, 并引入飞跃的特殊操作, 选择具有高但不太可能增益的设计点来避免局部极小值. 为了在充分利用并行计算基础设施的同时有效地加速探索过程, Mariani 等人[168] 提出 DeSpErate, 基于分布估计算法, 使用马尔可夫网估计帕累托最优解集的分布, 迭代地从分布中采样获取下一个采样点, 改进并行计算环境中的模拟计划, 而不是减少模拟的数量. DeSpErate++[26] 进一步用人工神经网络估计非支配级别对采样点进行排序, 以结构化的方式将人工神经网络和分布估算算法这 2 种技术结合在一起.

由于深度学习模型(包括深度强化学习、深度神经网络等)的卓越学习能力, 有研究基于此开展高效设计空间探索, 但目前主要用于领域专用设计架构. 利用深度强化学习方面, 针对 HLS 设计领域专用加

速器, IronMan[188] 利用图神经网络进行性能和资源预测, 设计基于强化学习的探索方法优化资源分配, 具体而言, 强化学习中将每个可能的部分分配数据流图作为状态, 将是否为数据流图节点分配制导语为动作, 将在用户指定的约束条件下对完全分配的数据流图的评估作为奖励函数, 采用 actor–critic 算法进行学习. 后续工作 IronMan-pro[189] 优化了图神经网络结构, 增加了策略梯度的学习方法, 设计了基于强化学习的多目标设计探索引擎优化资源分配策略, 旨在提供不同目标之间的帕累托解. ConfuciuX[190] 为给定的模型和数据流风格找到优化的硬件资源分配, 先使用基于 RNN 的策略网络强化学习探索执行单元和缓存的数量, 并使用遗传算法增强强化学习方法以进行微调. 利用深度神经网络方面, AIrchitect 通过定制的多层感知器(MLP)的神经网络来实现加速器设计空间探索, 其中网络输入是目标和解释神经网络结构的网络参数, 输出是加速器架构, 避免了候选设计的迭代搜索. 与 AIrchitect 相似, GANDSE[191] 对神经网络加速器设计自动化探索, 基于生成对抗网络(GAN)来优化高维设计空间的优化探索, 利用生成器和判别器迭代训练, 训练完成后在指定目标下直接给出最优设计.

6 模拟工具加速方法

本节总结利用模拟工具进行加速探索的相关工作, 包括软件模拟、硬件模拟和敏捷设计. 若将每个微架构设计点都进行完整的硅前评估, 则极为耗时而不现实, 无法满足处理器的到达市场时间, 如常用的基于软件的 RTL 模拟 Verilator 和 VCS 的速率只能达到 1~5 kHz, 仿真一个 SPEC2k6 负载需要约 5 年时间. 因此, 若干种模拟加速方法被提出. 软件模拟方法对设计点通过高抽象层次进行评估, 只关注微架构级事件周期的精确性而无需实现电路代码, 加速性能指标的评估. 这种方法牺牲准确性换取快速模拟和高灵活性. 此外, 通过 FPGA 可重复配置和高速并行运行的特性, 可将处理器的部分或全部逻辑通过硬件进行加速模拟, 模拟速度比软件模拟高 1 个数量级, 但开发难度较高, 且存在较高硬件成本开销. 随着现代设计工具的完善, 敏捷设计逐渐兴起, 可以快速进行微架构设计的 RTL 级电路实现并进行包含后端综合、布局布线等流程的性能评估. 敏捷设计更高的准确性且可接受的评估时间, 使得其可与高抽象层次的软件模拟互相补足. 表 10 对比了不同模拟工具的优劣势.

Table 10 Comparison of Simulation Tools

表 10 模拟工具的对比

类型	准确率	模拟速度	灵活性	开发难度
软件模拟	低	中(~10 MHz)	高	低
硬件模拟	中	快(~100 MHz)	低	中
敏捷设计	高	慢(1~5 kHz)	中	高

6.1 软件模拟

软件模拟指利用高级编程语言 C/C++/Python 等对架构及微架构进行行为模拟. 周期准确模拟器对微架构以周期精确或周期近似的方式进行模拟, 是使用最为广泛的一类软件模拟器. 此外, 功能模拟器指只模拟架构及指令集层次的功能, 一般用于分析负载微架构无关特征, 常通过二进制翻译和直接执行等技术, 将目标结构的指令通过翻译和直接执行的方式加速模拟, 但无法在微架构空间进行探索. 系统级模拟器常以事务级对微架构性能进行模拟, 关注重点常在 SoC 级的系统性能. 以下关注本综述涉及的加速方法工作中常用的周期准确模拟器. 更多模拟器的综合对比及综述可参见文献 [119, 192].

SimpleScalar[76] 是一个单核微架构模拟器, 由密歇根大学 Austin 团队提出, 支持包含快速模拟、分支预测器模拟、cache 模拟、乱序处理器模拟等多种模式, sim-outorder 模式下速率可达 500 KIPS. Manjikian[193] 扩展 SimpleScalar 以支持多核模式.

SESC(SuperEscalar Simulator)[77] 是 2005 年提出的周期准确模拟器, 支持单核处理器、CMP、存内计算(processing in memory, PIM)和线程级推测.

gem5[78] 是融合 GEMS 和 M5 模拟器的周期准确模拟器. M5 提供了一个高度可配置的仿真框架、多种指令集和多种 CPU 模型. GEMS 用一个详细的、可执行的内存系统来补充这些功能, 包括支持多种高速缓存一致性协议和互连模型. 目前, gem5 支持大多数商业指令集(ARM, ALPHA, MIPS, Power, SPARC, x86). Qureshi 等人[194] 后续将其扩展为多核异构平台 gem5-X, 研究了集群异构架构的设计空间探索.

Sniper[106] 由比利时汉德大学 Carlson 团队在 2011 年提出, 支持多核区间分析模型, 并在 Carlson 等人[195] 后续的机械模型工作中使用. Sniper 通过将精确的高度抽象分析模型与快速并行仿真相结合, 通过牺牲一定的精度换取模拟速度提升, 在一个 8 核对称多处理器上模拟一个 16 核系统时, 达到了 2.0 MIPS 的可扩展仿真速度, 模拟速度超过周期准确模拟器.

6.2 硬件模拟

利用 FPGA 或 Zebu 等硬件进行模拟加速的方法通过硬件的并行能力模拟处理器的功能模型或时序模型. 相关工作中, FAME[196] 基于 3 个维度进行分析, 包括直接/解耦、全 RTL/抽象机器、和单线程/多线程. 表 11 对比了不同硬件模拟平台.

Table 11 Comparison of Hardware Simulation Platforms

表 11 硬件模拟平台的对比

年份	平台	功能模拟	时序模拟	核心数	速率/(MIPS/核)
2007	FAST[79]	QEMU	FPGA	1	1.20
2009	ProtoFlex[80-81]	FPGA	软件	16	-
2010	RAMP Gold[82]	FPGA	FPGA	64	0.78
2011	HAsim[83]	FPGA	FPGA	15	8.47
2018	FireSim[84]	FPGA	FPGA	4 096	3.42

注：速率的单位为 MIPS/核. “-”表示文献无该数据.

FPGA 加速模拟技术具有一定的准确性, 并兼顾了运行速度, 其运行速度一般要比基于软件的时钟精准模拟器快 1 个数量级. 挑战在于, 相比传统的软件模拟, 开发 FPGA 模拟器的难度更高, 调试过程仅能依赖于底层的波形或电路信息, 没有能提供高层次信息等功能的标准库的基础设施. 此外, FPGA 综合及布局布线流程极为耗时, 且需要考虑 FPGA 运营成本.

FAST[79] 将模拟器划分为模拟指令集架构的推测性功能模型组件和预测性能的时序模型组件. 功能模型使用改进的全系统模拟器生成指令踪迹, 推测性功能模型使模拟器能够被并行化, 在 FPGA 中实现时序模型以提高速度. 其扩展性受限于 CPU 与 FPGA 之间的通信带宽.

PROTOFLEX[80-81] 只提供功能模拟的 FPGA 加速, 将许多逻辑处理器的执行虚拟到 FPGA 上的一组合并的多上下文执行引擎上, 降低了复杂性, 功能模拟速率达到 63 MIPS.

Tan 等人[82] 首先在 2010 年提出 RAMP Gold, 仿真 64 个顺序核心, 分离功能模型和时序模型. 在功能模型上利用 DSP 等资源加速流水线执行. 时序模型基于 SystemVerilog, 利用 BRAM 资源建模 cache, 但不支持片上网络的模拟. 文献 [196] 随后提出 FAME, 研究了硬件划分机制和操作系统调度策略之间的交互.

HAsim[83] 采用细粒度时分复用的方法模拟多核处理器及片上网络, 使用一块 FPGA 下单一的物理流水线建模多核时序, 模拟 4×4 的片上网络时, 单核模拟速率为 1.84 MHz, 多核模拟速率为 160 kHz.

相比于之前基于单个 FPGA 的模拟加速, FireSim[84] 运行在亚马逊 EC2 F1 公有云 FPGA 平台上, 支持面向集群服务器的微架构 RTL 级分布式并行仿真, 重点解决网络模拟的瓶颈, 可以以 3.4 MHz 的速度模拟 4 096 个核心.

6.3 敏捷设计

在过去的几年中, 敏捷芯片的开发和开源硬件受到越来越多的关注. 敏捷开发方法的目标是减少显著的工程成本和芯片开发的长设计周期, 其中快速生成处理器代码和评估的工作可以显著加速设计空间探索过程. 本节仅关注面向处理器设计的敏捷开发平台. 我们根据开发所使用语言抽象层次的高低将敏捷开发平台分为基于低级语言和基于高级语言 2 类.

表 12 对比了敏捷开发平台的所用语言类型、支持指令集、发布时间, 这些平台均提供 ASIC 和 FPGA 综合以及后端设计的能力. 从中可以看出, RISC-V 由于其开源的性质, 近年涌现了很多基于其的敏捷开发平台. 包云岗等人[17] 分析了敏捷处理器设计遇到的挑战, 并提出以面对对象体系结构设计方式为基础的处理器敏捷设计方法.

Table 12 Comparison of Agile Development Platforms

表 12 敏捷开发平台的对比

语言类型	平台	设计语言	指令集	年份
低级语言	OpenPiton[85]	Verilog HDL	SPARCv9	2016
	LiveHD[86]	Verilog HDL	RISC-V	2020
	BlackParrot [87]	SystemVerilog	RISC-V	2020
高级语言	CMD[88]	BlueSpec	RISC-V	2018
	Agile[197]	Chisel	RISC-V	2016
	Chipyard[89]	Chisel	RISC-V	2020
	MINJIE[50]	Chisel	RISC-V	2022
语言模型	llvm-mca[198]	-	-	2018
	Ithemal[199]	-	CISC	2019
	Chip-Chat[200]	自然语言	-	2023
	ChipGPT[201]	自然语言	RISC	2023
	RTLLM[202]	自然语言	RISC	2023

注: “-”表示文献无该项.

1)基于低级语言的平台

在电路设计实现方面, RTL 是目前使用最广泛的描述硬件的层次, 相关工作采用 Verilog HDL 和 SystemVerilog 进行开发. Balkind 等人[85] 开发了 OpenPiton, 它是一个用于构建多核的可扩展架构研究原型的开源框架, 通过 Python 提供简易的参数配置, 由

成熟的软件工具支持, 可运行全栈的多用户 Linux, 并由工业标准 Verilog 编写. 文献 [203] 讨论了 2015—2019 年这 5 年时间内开发过程中的经验教训和开源社区的发展历程. BlackParrot[87] 开源了 RISC-V 多核片上系统, 探索帕累托最优多核设计, 其开发工作优先考虑使用特制的接口和模块化以及硅验证作为首要的设计指标. LiveHD[86] 提出统一的超大规模集成电路数据模型 LGraph, 以支持综合和仿真的增量原则的实现.

2)基于高级语言的平台

由于基于 RTL 级的电路设计开发周期很长, 因此新兴出现了很多基于 RTL 但语言描述能力更为抽象的工作. Chisel[204] 是一种嵌入现代编程语言的新硬件描述语言, 并没有像高层次综合(high level synthesis, HLS)那样过度提高抽象水平, 而是创建电路生成器代替直接描述. Bluespec[205-206] 是一种高层次硬件描述语言, 其基于原子规则的规范的计算模型解决了多线程管理共享状态并发性容易出错的问题. 更多高级语言的工作可参见综述[207].

Lee 等人[197] 采用敏捷开发方法构建 11 个 RISC-V 处理器, 依赖于使用 Chisel 编写的硬件生成器对可制造的原型进行快速迭代改进. CMD[88] 是面向乱序核的可组合模块化设计框架, 采用 BlueSpec 开发, 提供模块的接口方法提供即时访问, 并对模块内的状态元素进行原子更新, 模块是由调用不同模块接口方法的原子规则组成的. Chipyard[89] 是集成 SoC 设计、仿真和实施环境, 包括可配置的、可组合的、开源的、基于生成器的知识产权块, 可以在硬件开发流程的多个阶段使用, 同时保持设计意图和集成一致性. 通过云托管的 FPGA 加速仿真和快速的 ASIC 实现, Chipyard 能够对物理上可实现的定制系统进行持续验证. MINJIE[50] 是面向高性能 RISC-V 处理器的一整套敏捷开发平台, 开源了包含除基于 Chisel 的处理器源码外, 还构建了包括仿真、验证、调试等在内的工具, 提出基于不同规则的敏捷验证方法, 基于此的香山处理器[208] 展示了该套平台的优越性. 随着越来越多高质量硬件项目采用敏捷开发工具, 敏捷开发方法未来在高性能和复杂设计领域将占据重要地位.

3)基于语言模型的平台

面向指令的语言模型可用于性能建模, 如 llvm-mca[198] 是以吞吐量和资源消耗对处理器性能进行分析的工具, 它使用 LLVM 中的可用信息(如调度模型)来静态测量机器代码在特定 CPU 中的性能. Ithemal[199] 无需指定 CPU, 其通过指令嵌入提取特征, 使用基于分层 LSTM 的模型预测指令块的吞吐量. 除了用于性能预测, 语言模型可用于生成硬件设计, 像 ChatGPT 这样的大型语言模型(large language model, LLM)表现出了前所未有的机器智能, 其在协助架构师通过自然语言交互实现更高效的逻辑设计方面也表现出了优越的性能. 如 Chip-Chat[200] 生成如基于 8 位累加器的微控制器的硬件设计, ChipGPT[201] 提出可扩展的 4 阶段零代码逻辑设计框架, 无需重新训练或微调. 除了关注设计的正确性, RTLLM[202] 同时考虑了设计质量, 提出了开源基准, 自动对任何给定的基于 LLM 的解决方案进行量化评估. 这类工作是实现硬件全自动化设计未来的一个重要里程碑.

7 性能模型加速方法

本节详细介绍性能模型(performance model). 性能模型对处理器的微架构与性能指标的关系进行建模, 常用于设计空间探索、运行时调度、负载瓶颈分析等任务中. 性能模型包括特定负载预测模型(prediction model)、跨负载预测模型、机械模型(mechanistic model). 预测模型, 也称经验模型(empirical model), 通过机器学习技术捕捉系统的行为, 以微架构参数为输入, 直接预测性能指标, 而不关注微架构内部组件, 是一种黑盒模型. 根据预测负载的使用场景, 我们将预测模型加速方法分为特定负载和跨负载 2 种类型. 机械模型基于对性能指标有影响的各种微架构事件进行分析, 常依赖缺失事件、依赖图的关键路径、概率统计信息等建模.

不同性能模型各有优劣, 如表 13 所示. 预测模型依靠一定量的训练数据即可对未评估的设计点进行较为准确的性能指标预测, 训练时间可接受且构建难度较低, 但它们并不能提供太多内部原因, 即可解释性较差, 适用于设计探索前期对巨大设计空间进行模糊探索阶段. 机械模型无需耗时的周期级模拟以获取训练数据, 常使用快 2 个量级的功能模拟获取相关统计数据, 允许通过比较模型公式中不同项的贡献来研究特别好的或坏的性能的来源, 常以 CPI 栈的形式分析性能瓶颈, 对性能的来源提供了更

Table 13 Comparison of Performance Models

表 13 性能模型的对比

类型	准确性	复杂度	可解释性
预测模型	低	低	低
机械模型	高	高	高

多的洞察，即可解释性较好，但需要针对特定的微架构进行调整，难以泛化到大规模的设计空间，构建难度较高，更适合于初步确定微架构设计后进行细致性能分析与参数微调.

7.1 特定负载预测模型

特定负载预测模型通过在只包含特定负载的性能指标数据集上训练，然后以微架构参数为输入，预测未评估过的设计点的性能指标. 我们可将性能指标聚合计算的负载集认为是一个新负载，如将测试基准 SPEC2k17 中负载的执行时间通过几何平均的方式进行聚合计算，此时可认为 SPEC2k17 为一个负载.

表 14 对比不同类型的性能预测模型，包括准确性、复杂度和可解释性. 表 15 对比特定负载预测模型. 各类模型的优缺点总结为：

Table 14 Comparison of Performance Prediction Models

表 14 性能预测模型的对比

类型	准确性	复杂度	可解释性
参数化	低	低	高
核函数	中	中	低
神经网络	中	高	低
树模型	中	中	高
集成学习	高	高	中

1）基于参数化的模型可以对参数及参数相关关系的重要性进行刻画，模型容易构建且可解释性很强，缺点在于泛化能力较其他模型弱.

2）基于核函数的模型通过将输入映射到高维空间，容易找到非线性相关的函数以刻画微架构参数-性能指标关系，构建难度较中等，难点在于模型性能

Table 15 Comparison of Workload-Specific Prediction Models

表 15 特定负载预测模型的对比

类型	预测模型	硬件设计空间	预测指标	负载	误差/%	R^2	采样/设计空间
参数化	线性回归[90]	单核	CPI	MinnerSPEC	0.8	-	$200/67\times10^6$
	受限三次样条回归[2,4]	单核、异构核	CPI, E, P	SPEC2k	4.9	-	$4\times10^3/22\times10^9$
	三次样条回归模型[5]	单核、多核	T	18 项负载	1.4	-	$300/4.3\times10^9$
	埃尔米特多项式插值[210]	PHT, cache	E	SPEC2k, MediaBench	-	-	$243/19\times10^3$
核函数	支持向量机[170]	单核	T, E	SPEC2k	0.5	-	12/4 608
	内核典型相关分析[211]	多核	T, E	ENePBench	6.2	0.88	$450/2.8\times10^6$
	ACOSSO[68]	单核、多核	T, E, P	SPEC2k, SPLASH-2	-	-	$450/128\times10^3$
	ACOSSO[13]	多核	T, E, P	SPLASH-2	-	-	$100/332\times10^3$
	高斯过程[91]	核数	T	SPLASH-3, PARSEC-3	-	0.82	67/68
	高斯过程[14]	单核	T, E, P	27 项负载	-	-	14/994
神经网络	径向基函数网络[92]	单核	CPI	MinnerSPEC	2.8	-	200/512
	小波神经网络[93]	单核	CPI, E, P	SPEC2k	-	-	$1\,024/246\times10^3$
	神经网络[3,209,212-213]	单核、多核	CPI	MinneSPEC 等	2.3	-	$221/23\times10^3$
	神经网络+遗传算法[214]	单核	CPI	SPEC2k	3.3		$230/23\times10^3$
树模型	模型树[94]	性能计数器	CPI	SPEC2k6	7.8	0.98	-
	模型树[95]	单核	T, E	图像压缩负载	1.3	0.95	3 211/3 288
	决策树[138]	性能计数器	CPI	SPEC2k6,SysMark07 等	2	-	-
	决策树[96]	异构核	T, E	SD-VBS, MiBench	2.1	-	664/830
集成学习	自适应提升+神经网络[67,97]	单核	CPI	SPEC2k6	-	-	$264/8.4\times10^6$
	梯度提升回归树[169]	单核、多核	T	SPEC2k, SPLASH-2	1.1	-	$3\times10^3/15\times10^6$
	XGBoost[172]	单核	E	riscv-tests	3.4	0.99	1 120/1 200
	提升法+梯度提升回归树[181]	单核	CPI, E, P	SPEC2k17	-	-	$100/2\times10^3$
	装袋法+模型树[98]	单核	CPI, E	SPEC2k	-	-	$320/71\times10^6$
	装袋法+梯度提升回归树[182]	单核	CPI, E, P	SPEC2k17	-	-	$100/37\times10^3$
	堆叠法+决策树[22]	单核、多核	T, E	SPEC2k6,SPLASH-2	-	-	$100/605\times10^3$
	堆叠法+异类模型[118]	单核	CPI, E	SPEC2k	1.8	-	$3\times10^3/2.5\times10^9$

注：硬件设计空间中单核主要包括单核处理器微架构，多核指基于总线或片上网络的同构多核处理器. “T”指时间，“E”指功率，“P”指对多个性能指标探索帕累托最优解集，误差以 CPI 的百分比绝对误差衡量（越接近 0 越好），R^2 为相关系数（越接近 1 越好），“-”表示该工作无显式标注数据.

受所选核函数影响较大,泛用性较差.

3)神经网络模型通过组合线性和非线性层学习规律,泛化能力中等且易于构建.缺点在于需要比其他模型多的样本进行训练,同时需要专业的算法知识调整模型以防止过拟合等问题,且可解释性较弱.

4)树模型的优势包括输入数据的灵活性以及在其预测变量中对条件信息建模的能力,适合于表格型数据和内插预测,预测能力中等.树模型在构建之后的结构是透明的,也就是说,设计者可以看到模型是如何得出预测的,从而洞察到变量的依赖关系和影响,可解释性较强,但缺点在于外推能力弱,预测未探索过的变量取值空间时准确度很低.

5)基于集成学习的模型在弱学习器的基础上进行强化,优点在于模型学习能力较强,缺点在于模型复杂度较高,需要专业算法知识调整模型,训练及预测的时间相比其他模型较长,可解释性中等.

此外,已有部分工作对比分析了特定负载预测模型的探索效果.文献 [91] 针对多线程加速比,对比了不同的机器学习模型,包括线性模型、k 近邻、支持向量机、随机森林、决策树和高斯过程模型,发现高斯过程模型效果最佳.针对多线程负载的性能,文献 [209] 强调分段多项式回归比神经网络提供了更好的可解释性,而神经网络显示出更好的泛化能力.MoDSE[181] 对比了若干种机器学习模型,实验表示集成树模型的效果最佳.

1)基于参数化的模型

基于参数化的模型常指经典的回归模型,通过学习权重参数确定回归函数的表达式.

EBLM[90] 利用 Akaike 的信息准则建立 CPI 的线性模型,重复迭代过程直到达到所需的误差界限.所得到的模型提供了所有微架构参数及其交互作用的显著性排序及方差.基于受限 3 次样条回归模型,Lee 等人[4] 预测单核 CPU 微架构的 CPI 和功率.基于该模型,文献 [2] 进一步探索了帕累托前沿、流水线深度,并进行多处理器异构性分析.文献 [5] 针对多核争用的情况,采用可组合性能回归(composable performance regression, CPR)构建 3 次样条回归模型预测单核 CPU 性能,在此基础上构建争用和惩罚模型估计多核性能.文献 [210] 采用三次埃尔米特多项式插值的局部回归模型对能耗进行建模,探索了分支预测器的模式历史表(pattern history table, PHT)和二级缓存层次的设计空间.

2)基于核函数的模型

基于核函数的模型通过将输入映射到高维空间,容易找到非线性相关的函数以刻画关系.

基于径向基核函数的支持向量机模型[170] 预测了性能与功率.文献 [211] 针对网络数据包处理负载,利用基于马氏距离的内核典型相关分析(kernel canonical correlation analysis, KCCA),寻找变量之间的最大相关性,以此预测片上多处理器的功耗和性能.文献 [13, 68] 采用自适应组件选择和平滑算子(ACOSSO)对执行周期数和功率分别建模.基于高斯过程模型,文献 [91] 预测多线程加速比,发现其在若干机器学习模型中表现最佳.BOOM-Explorer[14] 利用神经网络核函数的 2 任务高斯模型同时预测单核 CPU 的 CPI 和功率,并基于开源处理器核代码综合后的数据进行了实验,可信度比基于模拟器的工作要高,但测试基准的指令长度受评估速度限制.

3)基于神经网络的模型

神经网络通过堆叠多层线性函数与非线性激活函数进行训练,具备强大的拟合能力,且构建难度中等,因而广泛应用于性能预测模型中.

文献 [92] 采用非线性的径向基函数网络预测 CPI.同样基于径向基网络,文献 [93] 采用小波神经网络对 CPI、功率和可靠性指标进行建模.文献 [3] 采用一个隐藏层的人工神经网络预测 CPI.随后,文献 [209, 212–213] 利用相似的人工神经网络模型进行预测.文献 [10–11, 215–216] 将人工神经网络用于特定负载的性能预测模型,并随后将其内嵌于跨负载的性能预测模型.文献 [214] 构建神经网络模型,初始权值和学习常数由遗传算法优化.Özisikyilmaz 等人[217-218] 对比几种线性回归模型和不同的神经网络,表明修剪后的神经网络需要在较长训练时间的情况下获得最佳的精度,还利用模拟数据和真实处理器数据进行了验证.

4)基于树的模型

树模型将变量的特征和相应的阈值组织为节点的形式构建树结构进行学习和预测,包含模型树和决策树两大类.

利用模型树,文献 [94] 基于 M5P 算法识别负载中发现的独特的性能类别,并将每个类别与一个独特的、解释性能事件的线性模型联系起来.文献 [95] 预测了 FPGA 软核的配置和负载算法参数对图像压缩负载的性能和功耗的影响.

利用决策树,文献 [138] 通过 PCA 聚类识别最佳的代理负载集,学习性能计数器以预测 CPI.文献 [96] 计算特定于负载的比例因子以补偿预测的不准确性,主因子由模拟器和物理机之间的指标差异来测量,

次要因子由计算特征和存储特征的皮尔森相关系数来测量. McPAT-Calib[172] 利用 XGBoost 建模功率，自动选择适合的性能计数器，并采用综合后数据进行校准.

5）基于集成学习的模型

集成学习通过聚集多个基学习器的结果提升学习能力，按照集成的方法可分为 3 类：

①利用提升法. ActBoost[67] 和 SemiBoost[97] 对人工神经网络采用 AdaBoost. RT 集成算法，设置阈值过滤低置信度的神经网络预测值. 文献 [169] 基于梯度提升回归树，利用分类回归树作为基学习器，构建集成模型，选择最没信心（即预测方差值最大）的设计点，批量增加与该设计点超过距离阈值的设计点集. McPAT-Calib[172] 对比若干种模型，其中 XGBoost 的准确性最高. MoDSE[181] 通过实验最后使用自适应提升 GBRT 模型用于预测.

②利用装袋法. COMT[98] 设计协同训练模型树，初始化 2 个模型树，利用半监督方法，通过使用另一个模型标记的实例对每个模型进行细化. MoEnDSE[182] 设计装袋 GBRT 模型，利用方差估计不准确度进而迭代地进行探索.

③利用堆叠法. ArchRanker[22] 不是精确地估计设计点的性能，而是提出基于排序的方法，训练一个模型来预测 2 种体系结构配置中哪一种表现最好，其利用 RankBoost 算法训练决策树作为基学习器，再利用线性模型组合预测结果. ELSE[118] 构建了集成学习模型，包含模型树、神经网络、支持向量机，利用模型树作为元模型，小心维护模型的多样性，减少最终预测误差大到无法接受的情况，其中模型树准确度最高.

7.2 跨负载预测模型

跨负载（cross-workload）预测模型利用已收集到的源负载的性能数据，将知识迁移至目标负载空间，减少目标负载需要模拟的设计点数量. 跨负载预测模型分类包括基于负载特征、基于硬件响应、基于迁移学习的模型.

表 16 汇总了跨负载预测模型的工作，包括其使用的预测模型、跨负载方法核心、性能指标、训练集的设计点数（源负载的设计点数×源负载数量+目标负载的设计点数）、性能误差百分比、相关系数 R^2. 表 17 对比了 3 类跨负载预测模型的不同.

基于负载特征的模型首先考虑负载特征空间的相似性，仅使用微架构无关特征隐式或显式地增加模型的输入特征维度. 优势在于通过简便快速的功能模拟就可以刻画负载空间，构建难度较低. 不足在于损失了很多性能指标相关的知识，预测准确率较低.

基于硬件响应的模型直接依靠预测模型的学习能力，或将硬件响应做变换后作为负载特征的新增维度（称为签名）输入给预测模型. 优势是相比基于负载特征的模型，增加了性能指标相关的知识，提高了预测准确度. 但缺点在于需要额外的评估过程以获得签名，且签名的设计需要精心构建，构建难度中等.

基于迁移学习的模型对每个负载学习独立的预测模型，通过元模型的形式迁移源负载的知识，表现为 2 阶段的学习过程. 优势在于预测准确度较高. 不足在于学习过程复杂，基模型与元模型的训练耗时，

Table 16 Comparison of Cross-Workload Prediction Model Work

表 16 跨负载预测模型工作的对比

类型	来源	预测模型	跨负载方法核心	性能指标	设计点数	误差/%	R^2
负载特征	文献 [99]	归一化、PCA+GA、线性回归	负载特征、平均相似负载的结果	时间	35×25+0	-	-
	文献 [8]	多项式回归+遗传算法	负载特征	CPI	360×7+0	8~10	>0.90
	文献 [100]	模型树	负载特征	CPI、功率	500×25+0	-	0.90
	文献 [34]	多种模型之一	负载特征、最近邻归类模型	CPI、功率	3 000×10+0	-	0.98
硬件响应	文献 [9]	神经网络	模型本身泛化	时间	639×27+50	-	-
	文献 [23]	矩阵补全算法	模型本身泛化	CPI、功率	128×20+20	10.0	-
	文献 [101]	线性回归	响应边际关系、最近邻归类	CPI	60×23+600	6.3	0.92
	文献 [215]	神经网络	响应签名、模型本身泛化	时间、EDP	1 000×8+0	4.2	-
迁移学习	文献 [11]	神经网络	线性回归	CPI、功率	512×5+32	7.0	0.95
	文献 [10,216]	神经网络	贪心选择负载、线性回归	CPI、功率	512×5+32	3.0	-
	文献 [7]	模型树+自适应提升	负载聚类、样本迁移 TrAdaBoost	CPI	10×5+10	7.0	0.91
	文献 [6]	神经网络+自适应提升	支持向量机	CPI	128×3+40	5.5	0.93

注：“-”表示文献无该数据. “设计点数”列中的表达形式为源样本数量×源负载数量+目标样本数量.

Table 17 Comparison of Cross-Workload Prediction Models

表 17 跨负载预测模型的对比

类型	核心	准确性	复杂度
负载特征	特征空间的相似性	低	低
硬件响应	硬件响应作为新维度	中	中
迁移学习	元模型的知识迁移	高	高

且需要一定的源负载数据做支持.

1)基于负载特征的模型

基于负载特征的跨负载预测模型通过对微架构无关特征进行负载的刻画,基于负载的相似性进行聚类后,平均相似负载的结果选择部分特征训练预测模型,完成跨负载的预测.

文献 [99] 提出并评估了 3 种方法,包括归一化、主成分分析组合遗传算法、线性回归,将与微架构无关特征的原始数据集转换为基准空间,用欧氏距离评估相似性,以平均相似负载的性能指标为预测输出. 文献 [8] 将负载按照阶段收集微架构无关特征,构建多项式回归模型进行学习,利用遗传算法对软硬件参数的交互项进行自动化探索. CB[100] 利用相关性特征选择和遗传算法选取了部分微架构无关特征,将微架构无关特征和硬件配置作为输入来构建模型树预测模型. CASH[34] 先以微架构无关特征的距离评估相似性,再进行最近邻归类,利用类对应的最优模型模板对 CPI 和功率进行建模.

2)基于硬件响应的模型

基于硬件响应的模型常将负载在特定微架构集上的性能指标(也称为响应)作为签名,从负载的签名中挖掘相关性进行知识的迁移.

利用模型本身的泛化能力,文献 [9] 利用公开的性能数据库,采用神经网络模型对英特尔系列 CPU 在 SPEC2k6 和 Geekbench3 上的性能进行跨微架构、跨负载的预测. MPGM[23] 由性能指标构成稀疏矩阵,利用矩阵补全算法来跨负载预测,通过混合高斯模型增强采样分布中少见的部分. 文献 [101] 定义边际关系作为微架构参数变化引发的性能指标变化量,并将边际关系作为负载的相似度度量进行负载聚类,选择最接近的聚类的线性模型预测 CPI. RBM[215] 将在若干微架构参数下的能耗、延迟指标和微架构参数作为签名,输入给人工神经网络模型进行训练.

3)基于迁移学习的模型

基于迁移学习的模型对每个负载单独训练来预测模型,再利用元模型自动在负载之间迁移知识.

ANNLinear[11] 对每个负载训练一个人工神经网络模型,拼接若干在特定微架构上源负载的模型响应作为输入向量,基于线性回归预测目标负载的性能指标. EAC[216] 基于 ANNLinear,贪心选择若干最具代表性的源负载,再对性能指标进行线性回归,文献 [10] 增加能耗延迟平方指标的分析结果. TrEE[6] 利用 AdaBoost. ANN 作为基模型学习每个负载的 CPI,采用支持向量机用于元模型,执行基模型输出的非线性回归. TrDSE[7] 利用正交采样得到的性能指标分布,对负载进行层次聚类,选取与目标负载相似的源负载集,基于样本权重调整的迁移学习算法 TrAdaBoost. R2,将模型树作为基和元模型进行预测.

7.3 机械模型

本节总结机械模型,机械模型基于对性能指标有影响的各种微架构事件进行分析,不进行耗时的周期级模拟而快速获取性能指标. 我们按照分析方法将机械模型分类为分析模型、区间模型、图模型、概率统计模型、混合模型. 表 18 对比了不同文献所面向的目标架构、组件、预测指标、是否仅使用微架构无关特征、性能指标误差、速率. 各类机械模型之间的对比如表 19.

1)分析模型

在分析模型中,数学模型被用来定义各种负载属性(如指令混合和并行性)与不同微架构组件的相互作用. 与周期精确的模拟器不同,这些模型并不显式地模拟应用负载中的指令踪迹.

BCAT[219] 在预处理阶段,读取踪迹文件,构造一个二叉树数据结构,称为 2 分缓存分配树;主要处理阶段,计算缺失表,以 2 分缓存分配树在深度优先遍历;在后处理阶段,生成最优缓存配置,保证导致 cache 缺失率小于阈值. 文献 [102] 提出指令通量模型,该模型包括每个周期发出的组件指令与指令窗口大小之间的关系,由于分支错误预测,指令 cache 缺失、数据 cache 缺失和函数数量限制的惩罚. 关键思想是注意每个组件 IPC 的平均数量不能超过每个功能单元的专用数量.

CAMP[103] 估计由于 CMP 上的 cache 争用而导致性能下降,使用重用距离直方图、缓存访问频率以及每个进程的吞吐量和缓存失误率之间的关系来计算其并发运行和与其他进程共享缓存时的有效缓存大小,从而实现指令吞吐量估计. 文献 [130] 利用的是 L2cache 堆栈距离和循环序列剖析,提出了 3 个模型来预测缓存共享对共同调度线程的影响,包括访问频率模型、栈距离竞争模型和归纳概率模型. CACTI[220] 面

Table 18 Comparison of Mechanism Model Work

表 18 机械模型工作的对比

类型	来源	目标架构	组件	预测指标	仅微架构无关特征	误差/%	速率/(MIPS/核)
分析模型	文献 [219]	cache	cache	cache 缺失	✘	-	-
	文献 [102]	乱序、单核	指令窗口、BP、cache	IPC	✘	5.5	100
	文献 [103]	cache、多核	cache	IPC	✘	1.57	-
	文献 [220]	cache	cache	功率、面积	✘	5	-
	文献 [221]	按/乱序、多核	BP、cache、NoC 等	功率、面积	✘	11~23	-
区间模型	文献 [222]	乱序、单核	BP, cache	IPC	✘	5.8	-
	文献 [104]	乱序、单核	BP, cache	IPC	✘	7	-
	文献 [223]	乱序、多核	BP, cache	IPC	✘	4.6	~1
	文献 [105,224]	按序、单核	指令依赖、BP, cache	CPI、功率	✘	2.5	~6
	文献 [225]	乱序、单核	BP, cache	IPC、功率	✔	9.3	1.9
	文献 [226]	乱序、多核	BP, cache	CPI	✔	11.2	-
	文献 [227]	乱序、单核	SIMD、cache、带宽	CPI、功率	✔	25	-
	文献 [228]	乱序、多核	SIMD、cache、带宽	时间、功率	✔	36	-
图模型	文献 [107]	乱序、单核	BP, cache	CPI	✘	-	-
	文献 [108]	乱序、单核	BP, cache	CPI	✘	-	-
	文献 [109]	乱序、多核	BP, cache, NoC	IPC	✘	7.2	~12
概率统计模型	文献 [110]	乱序、单核	BP, cache	IPC	✘	2~10	-
	文献 [111]	乱序、多核	BP, cache	IPC	✘	7.9	~9
	文献 [112]	cache	cache	cache 缺失	✘	0.2	-
混合模型	文献 [113]	乱序、单核	流水线深度	IPC	✘	-	-
	文献 [114]	乱序、单核	cache、MSHR、预取	CPI、cache 缺失	✘	9.4	~15
	文献 [115]	乱序、单核	执行单元	CPI	✘	5.6	15.1

注: "-"表示文献无该数据.

Table 19 Comparison of Mechanism Models

表 19 机械模型的对比

类型	核心思想	准确性	复杂度
分析模型	数学公式	低	低
区间模型	事件分隔的区间	中	高
图模型	依赖图的关键路径	中	中
概率统计模型	事件发生的概率	中	中
混合模型	分析模型+预测模型	中	低

向 cache 和 SRAM 在内的基本存储单元, 通过已有的工艺数据库, 基于配置信息和读写访问统计数据公式化估计功耗、面积和时序. McPAT[221] 进一步整合了 CACTI, 通过分层次建模, 增加常见组合电路的数据库, 是一种被广泛使用的面向 CPU 在内的功耗、面积和时序估计工具.

2)区间模型

区间模型(interval model)[104] 提高了抽象级别, 用一个机械解析模型取代了核心级的周期精确模拟模型. 如图 4 所示, 区间模型将执行时间划分为由破坏性未命中事件(例如分支错误预测和缓存未命中)分隔的间隔. 每种类型的未命中事件都会导致执行时间间隔的可表征的性能行为. 通过考虑区间的类型和长度(以指令测量), 可以预测区间的执行时间. 然后通过聚合所有间隔的执行时间来确定总体执行时间.

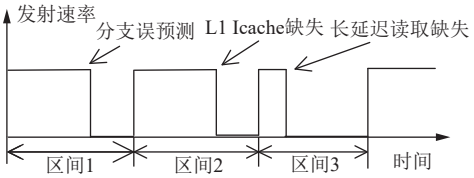

Fig. 4 Interval analysis analyzes performance on an interval basis determined by disruptive miss events[104]

图 4 区间分析对破坏性缺失事件确定的区间基性能进行了分析[104]

FOSPM[222] 最早提出区间建模, 建立了 1 阶乱序处理器模型, 对分支预测、指令和数据 cache 缺失对执行时间的影响进行了分析, 暂时不考虑处理器内有有限数量的功能单元或保留站的影响.

Eyerman 等人[104]提出单核超标量乱序处理器的区间模型, 分析了流水线深度和宽度的设计空间. 文献 [223] 进一步对多核处理器进行了建模, 分支预测器、内存层次结构、缓存一致性和互连网络模拟器决定了缺失事件; 解析模型推导出每个区间的时序. 机械模型与缺失事件模拟器之间的合作使多核处理器上的协同执行线程之间的紧密性能纠缠建模成为可能.

Carlson 等人[106]设计了 Sniper 模拟器实现多核区间分析模型. 文献 [195] 进一步引入了以指令窗口为中心的微架构模型, 通过支持具有更高细节级别的高速仿真, 弥合了区间仿真和周期精确仿真之间的差距.

Breughe 等人[224]针对按序超标量处理器, 对非单元指令执行延迟、指令间依赖关系、cache/TLB 缺失和分支错误预测的影响进行建模, 比较了按序与乱序处理器的性能, 量化了编译器优化对性能的影响. 文献 [105] 增加 ARM 处理器的验证, 确定实现给定性能目标的最佳功能单元数量; 分析负载-机器交互, 提供对微架构瓶颈的洞察; 分析编译器-架构交互、可视化编译器优化对性能的影响.

Jongerius 等人[227]分析单核处理器模型, 通过建模核间的缓存和内存带宽争用来支持多核系统的分析; 第 1 个包含 SIMD 的机械模型, 具有参数化的向量宽度; 根据指令类型单独考虑指令级并行性, 而不仅仅是其累积值, 以解释类型及其执行单元之间的不平衡. 随后, 文献 [228] 扩展到多核, 采用文献 [104] 的单核乱序机械模型和文献 [103] 模型来建模缓存竞争.

Van den Steen 等人[225]扩展了文献 [104], 利用微架构无关特征对 x86 乱序处理器建模, 仅需收集 1 次概要文件, 而无需对各个配置进行分析. 文献 [226] 进而提出 RPPM, 对多核处理器建模.

3)图模型

图模型将指令流水线构造为有向图, 通过计算关键路径获得性能指标. 如图 5 所示, 节点代表指令的流水线阶段, 连接节点的边表示流水线阶段之间的依赖关系, 边的权重表示操作的延迟. 通过找到从第一条指令的第 1 阶段到最后一条指令的最后阶段的最长路径(即关键路径), 可以得出执行所有指令所需的周期, 并确定这些周期花在哪里(即瓶颈). 但面对新的设计, 需要完全重新遍历图模型以找到新的关键路径. 考虑到典型的图和模拟大小(例如, 数百万个节点代表数百万条指令), 这将花费模拟器级

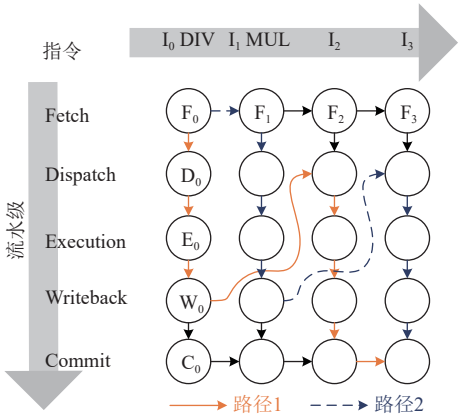

Fig. 5 A graph-based processor performance model[180]

图 5 基于图的处理器性能模型[180]

别的时间开销, 因此不能成为可扩展的解决方案, 仍是该方法面临的挑战.

CPADSE[107]使用负载模拟过程中产生的配置文件信息构建一个依赖图模型, 然后计算依赖图模型的关键路径, 并确定关键路径的模式, 提出模拟退火和禁忌搜索的设计空间探索方案. Lee 等人[108]提出 RpStacks, 其基于代表性的执行路径, 同时考虑了多个对性能有影响的行为, 以周期堆栈的形式提供了详细的性能瓶颈信息, 因此可以通过缩放堆栈的组件来即时估计一个图块的执行性能. 文献 [109] 随后提出 RpStacks-MT, 针对多核设计需要考虑的资源共享和动态调度, 设计基于重用距离的内存系统模型和一种动态调度重构方法.

4)概率统计模型

概率统计模型基于微架构事件发生的概率, 常构建马尔可夫链等模型对微架构状态及其跳转概率进行建模, 进而对性能指标进行预测. Noonburg 等人[110]提出了一个基于马尔可夫链的详细统计模型, 每个负载踪迹都被扫描一次, 生成一组负载并行度参数, 将机械模型和负载并行度参数结合起来形成马尔可夫链, 并且为了减少状态空间的大小, 对马尔可夫链进行了分区, 并使用迭代技术求解得到的链接模型. 文献 [111] 提出了一个马尔可夫链模型, 用于分析估计多核、细粒度多线程架构的吞吐量, 使用停止线程的数量作为状态, 并根据停止线程的事件的速率和延迟来计算转换概率. 通过对线程间停顿的重叠进行建模并考虑缓存争用. 基于马尔可夫的模型预测速度快, 但必须通过详细的分析为一组特定的处理器量身定做, 从而限制了该方法的灵活性. 除了马尔可夫链模型, 文献 [112] 面向指令 cache 微架构, 以概率的方式对控制流图中的每一个节点进

行 cache 状态的建模，探索相关 cache 配置的结构相似性，利用广义二叉树在一轮中估计多个配置的 cache 命中率.

5）混合模型

预测模型与机械模型的混合，也称灰盒建模，综合了 2 者的优点，但模型复杂度高. 文献 [113] 提出了一个以流水线深度为变量的性能公式，对最优流水线深度设计进行了探索. 从受机械建模启发的通用参数化性能模型开始，回归建模预测未知参数. 机械经验模型结合了 2 种模型的优点，提供机械模型的洞察力，同时如预测模型易于构建. 此外证明了混合机械经验模型比纯预测模型更稳健，更不易过拟合. 文献 [114] 预测挂起（pending）缓存命中、硬件预取和缺失状态保持寄存器（MSHR）资源对性能的影响，以 1 阶超标量机械模型为基础，根据个别负载的特点计算数据 cache 缺失的补偿系数. SimNet[115] 是以指令为中心的仿真框架，它将负载仿真分解为单个指令的延迟，并使用卷积神经网络进行指令延迟预测，包括取指延迟、执行延迟和存储延迟，通过结合所有执行指令的延迟预测结果来获得负载性能，利用图形处理器实现高并行度加速.

8 展　　望

本节对处理器微架构设计空间探索的加速方法未来研究的发展进行展望.

1）先进的机器学习算法

利用机器学习算法辅助处理器设计已经在学术界和产业界取得了一定的效果，而人工智能领域仍在蓬勃发展，随着更先进的算法被提出，必然可以推动设计空间探索方法的研究.

对于负载选择加速方法，过去基于降维和聚类的方法已达到一定的效果[29,31,35-37]. 如何进一步提升准确性是其面临的主要挑战，依靠于新的线性判别[38]和分组聚类[33]等算法目前显示出一定的提升潜力. 现有负载选择的方法并不能很好地覆盖如图计算和电子自动化设计等新领域负载的特性[37]. 如何在不同架构和微架构间保证负载选择的代表性是可待探索的领域，迁移学习和元学习[229-230]可以帮助探索新的选择方法，有望解决该类挑战.

对于设计点选择方法而言，统计推理方法[14,68]利用基于机器学习的代理模型和贝叶斯优化方法，针对特定的小硬件设计空间已达到较好的精度. 然而，面对更复杂的硬件设计空间如何提高搜索效率仍是重大挑战. 研究者可以关注新兴的人工智能算法，如主动学习、零知识迁移、大模型等，通过极少量数据或者其他领域的数据辅助空间的准确探索. 更好地利用领域知识，可以为不同的设计探索选择更合适的模型，并为这些模型的工作原因和方式提供更多的直觉或解释. 面对多目标优化问题如何进行权衡和避免局部最优解方面仍存在巨大的探索空间，例如强化学习[189]、对抗神经网络[191]等机器学习算法有希望优化探索结果.

性能模型方面，预测模型[3,14,21]通过训练机器学习模型实现从硬件微架构到性能指标的黑盒预测，构建效率较高，但精度受限于采样的设计点数量，且可解释性和迁移性较差. 面对下一代的设计空间，已有的性能数据面对新的设计时作用极为有限，如何保持数据的迁移灵活性是急需解决的问题. 在未来，随着集成学习[6,67]、主动学习[67,173]、迁移学习[6-7]等新出现的机器学习方法，研究者可进一步利用增强的性能模型的学习能力，例如通过人工智能大模型有可能引发准确性的颠覆式提升. 混合方法，即结合不同的机器学习技术或结合机器学习技术与启发式方法，将释放出更多的机会.

2）绝对转向相对

早期的设计空间探索过程重点在于绝对精度的评估，然而放松绝对精度的要求，追求相对精度则有望在保持探索结果可靠的情况下极大提升探索速度. 负载选择加速方法[29,35-37]通过挑选具备代表性的负载及其输入集减少了所要评估的负载耗时，而负载集上性能指标的相对提升趋势即可对微架构的升级做预测，有望进一步发掘加速潜能. 研究人员未来可关注通用性和准确性上的平衡，如借助高级描述语言等方法，保持相对精度而极大提升开发效率. 综合多层次的模拟结果也是一种利用相对精度的方法，如综合考虑高抽象层次、周期精确层次、电路实现层次的预测准确度和不确定性，通过校准[173]和微调等方式，有希望在有限的探索时间下进一步提升搜索效果[72,74].

3）并行负载的同步

随着多核同构甚至异构处理器的兴起，面对多线程负载的同步、资源竞争、负载调度等真实而复杂的情况时，性能指标的评估越发耗时，甚至带有明显的不确定性. 负载选择和部分模拟等方法在模型的复杂度急速攀升[34]下，存在较大的估计误差，可通过构建独立的内存模型、通信一致性模型等方式来提高访存模拟精度. 结合基于负载和同步的部分模

拟方法不失为一条可行之路. 此外, 考虑与性能模型配合的方式可以缩小不确定区间, 在提升准确性的情况下挖掘加速潜能. 贝叶斯优化[230]等方法将不确定性考虑进模型, 因此可以设计相关不确定感知的算法提升探索效果.

4) 敏捷开发

随着处理器设计复杂度的攀升, 开发和验证成本成为影响最终市场效果的关键因素, 敏捷开发工具应运而生. 为了更准确地评估执行时间、面积、功耗等性能指标, 需要支持后端综合及布局布线等流程工具的平台快速对处理器设计进行迭代评估, 将物理设计的性能参数评估引入探索过程. 将软件工程的思维引入敏捷开发是未来的发展趋势, 如包云岗等人[17]提出以面对对象体系结构设计方式为基础的处理器敏捷设计方法. 基于 RISC-V 的开源敏捷开发平台成为研究热点[50,87,203], 对产业界和工业界都带来了新机遇, 有望形成全新的体系结构设计流程. 然而, 验证和调试如何兼容不同层次是未来需要解决的问题, 目前 Zhang 等人[231]将软件源码级调试框架与从硬件生成器创建的 RTL 连接起来, 提升了调试效率. 此外, 处理器可靠性、安全、成本等新的衡量指标给模拟和设计工具带来了新的挑战, 如何进行快速而准确的评估是值得探索的问题, 建立起全栈式的模拟工具可达成事半功倍的效果. 基于硬件加速模拟已有一定的成效, 但过于复杂的构建过程和高昂的运维成本阻碍了大部分研究者, 研究者未来可采用云平台和众包开发等模式, 极大提升仿真速率和扩展性.

5) 领域专用加速器设计空间探索

随着近年领域专用加速器的火热, 良好的设计空间探索方法成为挖掘性能潜力的关键所在. 面向机器学习的 ASIC 加速器方面, 预测模型配合迭代搜索策略因其易用性将成为热门, 如 HASCO[232]采用多目标贝叶斯算法探索张量硬件设计, 并采用启发式算法和 Q-learning 强化学习优化软件, Esmaeizadeh 等人[233]对机器学习算法的加速器利用栈集成学习预测模型和 2 阶段推理进行设计探索. 基于 HLS 的 FPGA 加速器方面, Sun 等人[234]采用高斯模型与贝叶斯优化进行探索, IronMan-Pro[189]采用强化学习算法探索图神经网络加速器的设计. 由于领域专用加速器的执行机制对比处理器微架构较为简单, 而机械模型适合相关固定执行流程的设计, 且无需预测模型耗时的训练过程, 因此使用机械模型有希望进一步加速探索过程, 如 Sparseloop[235]针对稀疏张量加速器提出对稀疏度进行统计刻画, 同时考虑表示加速器特征. 此外, DNN 调度作为新考虑的设计维度, 涌现出许多探索性工作, 如 CoSA[236]通过建模为混合整数规划问题进行探索, ZigZag[237]利用不均匀的映射和启发式搜索, 都试图通过软硬件协同的方式充分挖掘探索效果, 这是研究者未来可重点挖掘的方向.

未来针对新型领域加速器的设计挑战将会持续上升, 而设计空间探索方法愈发重要. 首先, 新的设计空间需要考虑协同软硬件设计, 现在的诸多模板化加速器设计框架仍与特定领域紧耦合, 对更复杂的云端、边端、物端场景约束, 在专用性和通用性上如何平衡亟待解决. 其次, 面对任务划分和映射等复杂的设计空间, 如何跨层融合进行探索, 需要更智能的搜索方法及更准确的性能模型, 如混合预测模型和机械模型. 最后, 领域专用编译器等工具在提升效率的同时, 应用层到硬件层之间存在表达语义的鸿沟, 如何全栈式自动探索设计是未来的一个发展方向.

9 结 论

本文对处理器微架构设计空间探索的加速方法进行系统总结及分类, 包含软件设计空间的负载选择、负载指令的部分模拟、设计点选择、模拟工具、和性能模型 5 类加速方法. 本文对比了各加速方法内文献之间的异同, 覆盖了从软件选择到硬件设计的完整探索流程. 我们相信本文对处理器微架构设计空间探索加速方法的系统性总结和前沿研究方向的展望将有助于读者了解微架构设计空间探索的研究现状和挖掘未来研究方向.

作者贡献声明: 王铎负责文献综述和数据分析等工作; 王铎、刘景磊负责背景、设计点选择、模拟加速等章节的文献整理及撰写; 王铎、严明玉、滕亦涵、韩登科负责论文逻辑组织、内容梳理及撰写等工作; 叶笑春、范东睿、刘景磊负责研究思路梳理及研究路线设计等工作.

参 考 文 献

[1] Azizi O, Mahesri A, Lee B C, et al. Energy-performance tradeoffs in processor architecture and circuit design: A marginal cost analysis[C]//Proc of the 27th Annual Int Symp on Computer Architecture. New York: ACM, 2010: 26–36

[2] Lee B C, Brooks D M. Illustrative design space studies with microarchitectural regression models[C]//Proc of the 13th Int Conf

on High-Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2007: 340–351

[3] Ipek E, McKee S A, Caruana R, et al. Efficiently exploring architectural design spaces via predictive modeling[C]//Proc of the 12th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2006: 195–206

[4] Lee B C, Brooks D M. Accurate and efficient regression modeling for microarchitectural performance and power prediction[C]//Proc of the 12th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2006: 185–194

[5] Lee B C, Collins J D, Wang Hong, et al. CPR: Composable performance regression for scalable multiprocessor models[C]//Proc of the 41st Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2008: 270–281

[6] Li Dandan, Yao Shuzhen, Wang Senzhang, et al. Cross-program design space exploration by ensemble transfer learning[C]//Proc of the 36th IEEE/ACM Int Conf on Computer-Aided Design. Piscataway, NJ: IEEE, 2017: 201–208

[7] Li Dandan, Wang Senzhang, Yao Shuzhen, et al. Efficient design space exploration by knowledge transfer[C]//Proc of the 11th IEEE/ACM/IFIP Int Conf on Hardware/Software Codesign and System Synthesis. New York: ACM, 2016: 12: 1−12: 10

[8] Wu Weidan, Lee B C. Inferred models for dynamic and sparse hardware-software spaces[C]//Proc of the 45th Annual IEEE/ACM Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 2012: 413–424

[9] Wang Yu, Lee V, Wei G Y, et al. Predicting new workload or CPU performance by analyzing public datasets[J]. ACM Transactions on Architecture and Code Optimization, 2019, 15(4): 53: 1−53: 21

[10] Dubach C, Jones T M, O'Boyle M F P. An empirical architecture-centric approach to microarchitectural design space exploration[J]. IEEE Transactions on Computers, 2011, 60(10): 1445−1458

[11] Dubach C, Jones T M, O'Boyle M F P. Microarchitectural design space exploration using an architecture-centric approach[C]//Proc of the 40th Annual IEEE/ACM Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 2007: 262–271

[12] Eeckhout L, De Bosschere K. Speeding up architectural simulations for high-performance processors[J]. Simulation, 2004, 80(9): 451−468

[13] Wang Hongwei, Shi Jinglin, Zhu Ziyuan. An expected hypervolume improvement algorithm for architectural exploration of embedded processors[C]//Proc of the 53rd Annual Design Automation Conf. New York: ACM, 2016: 161: 1−161: 6

[14] Bai Chen, Sun Qi, Zhai Jianwang, et al. BOOM-Explorer: RISC-V BOOM microarchitecture design space exploration framework[C/OL]//Proc of the 40th IEEE/ACM Int Conf on Computer Aided Design. Piscataway, NJ: IEEE, 2021[2023-12-17]. https://ieeexplore.ieee.org/document/9643455

[15] Yi J J, Lilja D J, Hawkins D M. A statistically rigorous approach for improving simulation methodology[C]//Proc of the 9th Int Symp on High-Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2003: 281–291

[16] Monchiero M, Canal R, González A. Power/performance/thermal design-space exploration for multicore architectures[J]. IEEE Transactions on Parallel and Distributed Systems, 2008, 19(5): 666−681

[17] Bao Yungang, Chang Yisong, Han Yinhe, et al. Agile design of processor chips: Issues and challenges[J]. Journal of Computer Research and Development, 2021, 58(6): 1131−1145 (in Chinese)
(包云岗, 常轶松, 韩银和, 等. 处理器芯片敏捷设计方法: 问题与挑战[J]. 计算机研究与发展, 2021, 58(6): 1131−1145)

[18] Standard Performance Evaluation Corporation. SPEC CPU2017[EB/OL]. (2012-12-06)[2023-12-01]. https://www.spec.org/cpu2017

[19] Yi J J, Lilja D J. Simulation of computer architectures: Simulators, benchmarks, methodologies, and recommendations[J]. IEEE Transactions on Computers, 2006, 55(3): 268−280

[20] Guo Qi, Chen Tianshi, Chen Yunji, et al. Accelerating architectural simulation via statistical techniques: A survey[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2016, 35(3): 433−446

[21] O'Neal K, Brisk P. Predictive modeling for CPU, GPU, and FPGA performance and power consumption: A survey[C]//Proc of the 2018 IEEE Computer Society Annual Symp on VLSI. Los Alamitos, CA: IEEE Computer Society, 2018: 763–768

[22] Chen Tianshi, Guo Qi, Tang Ke, et al. ArchRanker: A ranking approach to design space exploration[C]//Proc of the 41st Int Symp on Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2014: 85–96

[23] Ding Yi, Mishra N, Hoffmann H. Generative and multi-phase learning for computer systems optimization[C]//Proc of the 46th Int Symp on Computer Architecture. New York: ACM, 2019: 39–52

[24] Panerati J, Beltrame G. A comparative evaluation of multi-objective exploration algorithms for high-level design[J]. ACM Transactions on Design Automation of Electronic Systems, 2014, 19(2): 15: 1–15: 22

[25] Palermo G, Silvano C, Zaccaria V. ReSPIR: A response surface-based pareto iterative refinement for application-specific design space exploration[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2009, 28(12): 1816−1829

[26] Mariani G, Palermo G, Zaccaria V, et al. DeSpErate++: An enhanced design space exploration framework using predictive simulation scheduling[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, 34(2): 293−306

[27] Cammarota R, Beni L A, Nicolau A, et al. Effective evaluation of multi-core based systems[C]//Proc of the 12th Int Symp on Parallel and Distributed Computing. Piscataway, NJ: IEEE, 2013: 19–25

[28] KleinOsowski A J, Lilja D J. MinneSPEC: A new spec benchmark workload for simulation-based computer architecture research[J]. IEEE Computer Architecture Letters, 2002, 1(1): 7−10

[29] Eeckhout L, Vandierendonck H, De Bosschere K. Workload design: Selecting representative program-input pairs[C]//Proc of the 11th Int Conf on Parallel Architectures and Compilation Techniques. Los Alamitos, CA: IEEE Computer Society, 2002: 83–94

[30] Breughe M, Eeckhout L. Selecting representative benchmark inputs for exploring microprocessor design spaces[J]. ACM Transactions on Architecture and Code Optimization, 2013, 10(4): 37: 1−37: 24

[31] Joshi A, Phansalkar A, Eeckhout L, et al. Measuring benchmark similarity using inherent program characteristics[J]. IEEE

Transactions on Computers, 2006, 55(6): 769−782

[32] Vandeputte F, Eeckhout L. Phase complexity surfaces: Characterizing time-varying program behavior[C]//Proc of the 3rd High Performance Embedded Architectures and Compilers. Berlin: Springer, 2008: 320–334

[33] Zhan Hongping, Lin Weiwei, Mao Feiqiao, et al. BenchSubset: A framework for selecting benchmark subsets based on consensus clustering[J]. International Journal of Intelligent Systems, 2022, 37(8): 5248−5271

[34] Sheidaeian H, Fatemi O. Toward a general framework for jointly processor-workload empirical modeling[J]. The Journal of Supercomputing, 2021, 77(6): 5319−5353

[35] Phansalkar A, Joshi A, John L K. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite[C]//Proc of the 34th Int Symp on Computer Architecture. New York: ACM, 2007: 412–423

[36] Limaye A, Adegbija T. A workload characterization of the SPEC CPU2017 benchmark suite[C]//Proc of the 2018 IEEE Int Symp on Performance Analysis of Systems and Software. Los Alamitos, CA: IEEE Computer Society, 2018: 149–158

[37] Panda R, Song Shuang, Dean J, et al. Wait of a decade: Did SPEC CPU 2017 broaden the performance horizon[C]//Proc of the 23rd IEEE Int Symp on High Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2018: 271–282

[38] Liu Qingrui, Wu Xiaolong, Kittinger L, et al. BenchPrime: Effective building of a hybrid benchmark suite[J]. ACM Transactions in Embedded Computing Systems, 2017, 16(5): 179: 1−179: 22

[39] Wunderlich R E, Wenisch T F, Falsafi B, et al. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling[C]//Proc of the 30th Annual Int Symp on Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2003: 84–95

[40] Hassani S, Southern G, Renau J. LiveSim: Going live with microarchitecture simulation[C]//Proc of the 22nd IEEE Int Symp on High Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2016: 606–617

[41] Hamerly G, Perelman E, Lau J, et al. SimPoint 3.0: Faster and more flexible program phase analysis[J/OL]. Journal of Instruction-Level Parallelism, 2005[2023-12-18]. http://www.jilp.org/vol7/v7paper14.pdf

[42] Sherwood T, Perelman E, Hamerly G, et al. Discovering and exploiting program phases[J]. IEEE Micro, 2003, 23(6): 84−93

[43] Shen Xipeng, Zhong Yutao, Ding Chen. Locality phase prediction[C]//Proc of the 11th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2004: 165–176

[44] Ardestani E K, Renau J. ESESC: A fast multicore simulator using time-based sampling[C]//Proc of the 19th Int Symp on High Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2013: 448–459

[45] Jiang Chuntao, Yu Zhibin, Jin Hai, et al. PCantorSim: Accelerating parallel architecture simulation through fractal-based sampling[J]. ACM Transactions on Architecture and Code Optimization, 2013, 10(4): 49: 1–49: 24

[46] Sabu A, Patil H, Heirman W, et al. LoopPoint: Checkpoint-driven sampled simulation for multi-threaded applications[C]//Proc of the 28th Int Symp on High-Performance Computer Architecture. Piscataway, NJ: IEEE, 2022: 604–618

[47] Carlson T E, Heirman W, Van Craeynest K, et al. BarrierPoint: Sampled simulation of multi-threaded applications[C]//Proc of the 2014 IEEE Int Symp on Performance Analysis of Systems and Software. Los Alamitos, CA: IEEE Computer Society, 2014: 2–12

[48] Grass T, Carlson T E, Rico A, et al. Sampled simulation of task-based programs[J]. IEEE Transactions on Computers, 2019, 68(2): 255−269

[49] Wenisch T F, Wunderlich R E, Ferdman M, et al. SimFlex: Statistical sampling of computer system simulation[J]. IEEE Micro, 2006, 26(4): 18−31

[50] Xu Yinan, Yu Zihao, Tang Dan, et al. Towards developing high performance RISC-V processors using agile methodology[C]//Proc of the 55th IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2022: 1178–1199

[51] Bryan P D, Rosier M C, Conte T M. Reverse state reconstruction for sampled microarchitectural simulation[C]//Proc of the 2007 IEEE Int Symp on Performance Analysis of Systems & Software. Los Alamitos, CA: IEEE Computer Society, 2007: 190–199

[52] Nussbaum S, Smith J E. Modeling superscalar processors via statistical simulation[C]//Proc of the 10th Int Conf on Parallel Architectures and Compilation Techniques. Los Alamitos, CA: IEEE Computer Society, 2001: 15–24

[53] Eeckhout L, Nussbaum S, Smith J E, et al. Statistical simulation: Adding efficiency to the computer designer's toolbox[J]. IEEE Micro, 2003, 23(5): 26−38

[54] Oskin M, Chong F T, Farrens M. HLS: Combining statistical and symbolic simulation to guide microprocessor designs[C]//Proc of the 27th Int Symp on Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2000: 71–82

[55] Bell R H, John L K. Improved automatic testcase synthesis for performance model validation[C]//Proc of the 19th Annual Int Conf on Supercomputing. New York: ACM, 2000: 111–120

[56] Genbrugge D, Eeckhout L. Statistical simulation of chip multiprocessors running multi-program workloads[C]//Proc of the 25th Int Conf on Computer Design. Piscataway, NJ: IEEE, 2007: 464–471

[57] Genbrugge D, Eeckhout L. Chip multiprocessor design space exploration through statistical simulation[J]. IEEE Transactions on Computers, 2009, 58(12): 1668−1681

[58] Hughes C, Li T. Accelerating multi-core processor design space evaluation using automatic multi-threaded workload synthesis[C]// Proc of the 4th Int Symp on Workload Characterization. Los Alamitos, CA: IEEE Computer Society, 2008: 163–172

[59] Balakrishnan G, Solihin Y. WEST: Cloning data cache behavior using stochastic traces[C]//Proc of the 18th IEEE Int Symp on High-Performance Comp Architecture. Los Alamitos, CA: IEEE Computer Society, 2012: 1–12

[60] Awad A, Solihin Y. STM: Cloning the spatial and temporal memory access behavior[C]//Proc of the 20th IEEE Int Symp on High Performance Computer Architecture. Los Alamitos, CA: IEEE

Computer Society, 2014: 237–247

[61] Wang Yipeng, Awad A, Solihin Y. Clone morphing: Creating new workload behavior from existing applications[C]//Proc of the 2017 IEEE Int Symp on Performance Analysis of Systems and Software. Los Alamitos, CA: IEEE Computer Society, 2017: 97–108

[62] Wang Yipeng, Balakrishnan G, Solihin Y. MeToo: Stochastic modeling of memory traffic timing behavior[C]//Proc of the 24th Int Conf on Parallel Architecture and Compilation. Los Alamitos, CA: IEEE Computer Society, 2015: 457–467

[63] Hekstra G J, La Hei G D, Bingley P, et al. TriMedia CPU64 design space exploration[C]//Proc of the 17th IEEE Int Conf on Computer Design: VLSI in Computers and Processors. Los Alamitos, CA: IEEE Computer Society, 1999: 599–606

[64] Fornaciari W, Sciuto D, Silvano C, et al. A design framework to efficiently explore energy-delay tradeoffs[C]//Proc of the 9th Int Symp on Hardware/Software Codesign. New York: ACM, 2001: 260–265

[65] Fornaciari W, Sciuto D, Silvano C, et al. A sensitivity-based design space exploration methodology for embedded systems[J]. Design Automation for Embedded Systems, 2002, 7(1): 7−33

[66] Sheldon D, Kumar R, Lysecky R, et al. Application-specific customization of parameterized FPGA soft-core processors[C]//Proc of the 25th IEEE/ACM Int Conf on Computer-Aided Design. New York: ACM, 2006: 261–268

[67] Li Dandan, Yao Shuzhen, Liu Yuhang, et al. Efficient design space exploration via statistical sampling and AdaBoost learning[C]//Proc of the 53rd Annual Design Automation Conf. New York: ACM, 2016: 142: 1−142: 6

[68] Wang Hongwei, Zhu Ziyuan, Shi Jinglin, et al. An accurate acosso metamodeling technique for processor architecture design space exploration[C]//Proc of the 20th Asia and South Pacific Design Automation Conf. Piscataway, NJ: IEEE, 2015: 689–694

[69] Mariani G, Palermo G, Zaccaria V, et al. Design-space exploration and runtime resource management for multicores[J]. ACM Transactions on Embedded Computing Systems, 2013, 13(2): 20: 1−20: 27

[70] Jahr R, Calborean H, Vintan L, et al. Boosting design space explorations with existing or automatically learned knowledge[C]// Proc of the 15th Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance. Berlin: Springer, 2012: 221–235

[71] Palesi M, Givargis T. Multi-objective design space exploration using genetic algorithms[C]//Proc of the 10th Int Symp on Hardware/ Software Codesign. New York: ACM, 2002: 67–72

[72] Eyerman S, Eeckhout L, De Bosschere K. Efficient design space exploration of high performance embedded out-of-order processors[C]//Proc of the 9th Design, Automation & Test in Europe Conf and Exhibition. Piscataway, NJ: IEEE, 2006: 351−356

[73] Ascia G, Catania V, Di Nuovo A G, et al. Efficient design space exploration for application specific systems-on-a-chip[J]. Journal of Systems Architecture, 2007, 53(10): 733−750

[74] Mariani G, Palermo G, Silvano C, et al. Multi-processor system-on-chip design space exploration based on multi-level modeling techniques[C]//Proc of the 9th Int Conf on Embedded Computer Systems: Architectures, Modeling and Simulation. Piscataway, NJ: IEEE, 2009: 118–124

[75] Mariani G, Palermo G, Zaccaria V, et al. OSCAR: An optimization methodology exploiting spatial correlation in multicore design spaces[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2012, 31(5): 740−753

[76] Burger D, Austin T M. The SimpleScalar tool set, version 2.0[J]. ACM SIGARCH Computer Architecture News, 1997, 25(3): 13−25

[77] Renau J, Fraguela B, Tuck J, et al. SESC simulator[EB/OL]. 2005[2023-12-01]. http://sesc.sourceforge.net

[78] Binkert N, Beckmann B, Black G, et al. The gem5 simulator[J]. ACM SIGARCH Computer Architecture News, 2011, 39(2): 1−7

[79] Chiou D, Sunwoo D, Kim J, et al. FPGA-accelerated simulation technologies (FAST): Fast, full-system, cycle-accurate simulators[C]//Proc of the 40th Annual IEEE/ACM Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 2007: 249–261

[80] Chung E S, Nurvitadhi E, Hoe J C, et al. A complexity-effective architecture for accelerating full-system multiprocessor simulations using FPGAs[C]//Proc of the 16th Int ACM/SIGDA Symp on Field Programmable Gate Arrays. New York: ACM, 2008: 77–86

[81] Chung E S, Papamichael M K, Nurvitadhi E, et al. ProtoFlex: Towards scalable, full-system multiprocessor simulations using FPGAs[J]. ACM Transactions on Reconfigurable Technology and Systems, 2009, 2(2): 15: 1–15: 32

[82] Tan Zhangxi, Waterman A, Avizienis R, et al. RAMP Gold: An FPGA-based architecture simulator for multiprocessors[C]//Proc of the 47th Design Automation Conf. New York: ACM, 2010: 463–468

[83] Pellauer M, Adler M, Kinsy M, et al. HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing[C]// Proc of the 17th Int Symp on High Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2011: 406–417

[84] Karandikar S, Mao H, Kim D, et al. FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud[C]//Proc of the 45th Annual Int Symp on Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2018: 29–42

[85] Balkind J, McKeown M, Fu Yaosheng, et al. OpenPiton: An open source manycore research framework[C]//Proc of the 21st Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2016: 217–232

[86] Wang Shenghong, Possignolo R T, Skinner H B, et al. LiveHD: A productive live hardware development flow[J]. IEEE Micro, 2020, 40(4): 67−75

[87] Petrisko D, Gilani F, Wyse M, et al. BlackParrot: An agile open-source RISC-V multicore for accelerator socs[J]. IEEE Micro, 2020, 40(4): 93−102

[88] Zhang Sizhuo, Wright A, Bourgeat T, et al. Composable building blocks to open up processor design[C]//Proc of the 51st Annual IEEE/ACM Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 2018: 68–81

[89] Amid A, Biancolin D, Gonzalez A, et al. Chipyard: Integrated design, simulation, and implementation framework for custom socs[J]. IEEE Micro, 2020, 40(4): 10−21

[90] Joseph P J, Vaswani K, Thazhuthaveetil M J. Construction and use of linear regression models for processor performance analysis[C]// Proc of the 12th Int Symp on High-Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2006: 99–108

[91] Agarwal N, Jain T, Zahran M. Performance prediction for multi-threaded applications[C]//Proc of the 2nd Int Workshop on AI-assisted Design for Architecture. New York: ACM, 2019: 71−76

[92] Joseph P J, Vaswani K, Thazhuthaveetil M J. A predictive performance model for superscalar processors[C]//Proc of the 39th Annual IEEE/ACM Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 2006: 161–170

[93] Cho C B, Zhang Wangyuan, Li Tao. Informed microarchitecture design space exploration using workload dynamics[C]//Proc of the 40th Annual IEEE/ACM Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 2007: 274–285

[94] Ould-Ahmed-Vall E, Woodlee J, Yount C, et al. Using model trees for computer architecture performance analysis of software applications[C]//Proc of the 2007 IEEE Int Symp on Performance Analysis of Systems and Software. Los Alamitos, CA: IEEE Computer Society, 2007: 116–125

[95] Powell A, Savvas-Bouganis C, Cheung P Y K. High-level power and performance estimation of FPGA-based soft processors and its application to design space exploration[J]. Journal of Systems Architecture, 2013, 59(10): 1144−1156

[96] Mankodi A, Bhatt A, Chaudhury B. Predicting physical computer systems performance and power from simulation systems using machine learning model[J]. Computing, 2022, 105(5): 1−19

[97] Li Dandan, Yao Shuzhen, Wang Ying. Processor design space exploration via statistical sampling and semi-supervised ensemble learning[J]. IEEE Access, 2018, 6: 25495−25505

[98] Guo Qi, Chen Tianshi, Chen Yunji, et al. Effective and efficient microprocessor design space exploration using unlabeled design configurations[C]//Proc of the 22nd Int Joint Conf on Artificial Intelligence. Palo Alto, CA: AAAI, 2011: 1671–1677

[99] Hoste K, Phansalkar A, Eeckhout L, et al. Performance prediction based on inherent program similarity[C]//Proc of the 15th Int Conf on Parallel Architectures and Compilation Techniques. New York: ACM, 2006: 114–122

[100] Guo Qi, Chen Tianshi, Chen Yunji, et al. Microarchitectural design space exploration made fast[J]. Microprocessors and Microsystems, 2013, 37(1): 41−51

[101] Ahmadinejad H, Fatemi O. Moving towards grey-box predictive models at micro-architecture level by investigating inherent program characteristics[J]. IET Computers Digital Techniques, 2018, 12(2): 53−61

[102] Taha T M, Wills S. An instruction throughput model of superscalar processors[J]. IEEE Transactions on Computers, IEEE, 2008, 57(3): 389−403

[103] Xu Chi, Chen Xi, Dick R P, et al. Cache contention and application performance prediction for multi-core systems[C]//Proc of the 2010 IEEE Int Symp on Performance Analysis of Systems & Software. Los Alamitos, CA: IEEE Computer Society, 2010: 76–86

[104] Eyerman S, Eeckhout L, Karkhanis T, et al. A mechanistic performance model for superscalar out-of-order processors[J]. ACM Transactions on Computer Systems, 2009, 27(2): 3: 1–3: 37

[105] Breughe M B, Eyerman S, Eeckhout L. Mechanistic analytical modeling of superscalar in-order processor performance[J]. ACM Transactions on Architecture and Code Optimization, 2015, 11(4): 50: 1–50: 26

[106] Carlson T E, Heirman W, Eeckhout L. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation[C]//Proc of the 2011 Conf on High Performance Computing Networking, Storage and Analysis. New York: ACM, 2011: 52: 1−52: 12

[107] Wang Lei, Tang Yuxing, Deng Yu, et al. A scalable and fast microprocessor design space exploration methodology[C]//Proc of the 9th Int Symp on Embedded Multicore/Many-core Systems-on-Chip. Los Alamitos, CA: IEEE Computer Society, 2015: 33–40

[108] Lee J, Jang H, Kim J. RpStacks: Fast and accurate processor design space exploration using representative stall-event stacks[C]//Proc of the 47th Annual IEEE/ACM Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 2014: 255–267

[109] Jang H, Jo J E, Lee J, et al. RpStacks-MT: A high-throughput design evaluation methodology for multi-core processors[C]//Proc of the 51st Annual IEEE/ACM Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 2018: 586–599

[110] Noonburg D B, Shen J P. A framework for statistical modeling of superscalar processor performance[C]//Proc of the 3rd Int Symp on High-Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 1997: 298–309

[111] Chen X E, Aamodt T M. A first-order fine-grained multithreaded throughput model[C]//Proc of the 15th Int Symp on High Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2009: 329–340

[112] Liang Y, Mitra T. An analytical approach for fast and accurate design space exploration of instruction caches[J]. ACM Transactions on Embedded Computing Systems, 2013, 13(3): 43: 1−43: 29

[113] Hartstein A, Puzak T R. The optimum pipeline depth for a microprocessor[C]//Proc of the 29th Annual Int Symp on Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2002: 7–13

[114] Chen X E, Aamodt T M. Hybrid analytical modeling of pending cache hits, data prefetching, and mshrs[J]. ACM Transactions on Architecture and Code Optimization, 2011, 8(3): 59−70

[115] Li L, Pandey S, Flynn T, et al. SimNet: Accurate and high-performance computer architecture simulation using deep learning[C]//Proc of the 2022 ACM SIGMETRICS/IFIP Performance Joint Int Conf on Measurement and Modeling of Computer Systems. New York: ACM, 2022: 67–68

[116] Panda R, John L K. Proxy benchmarks for emerging big-data workloads[C]//Proc of the 26th Int Conf on Parallel Architectures and Compilation Techniques. Los Alamitos, CA: IEEE Computer Society, 2017: 105–116

[117] Kang S, Kumar R. Magellan: A search and machine learning-based framework for fast multi-core design space exploration and optimization[C]//Proc of the 2008 Design, Automation and Test in Europe. New York: ACM, 2008: 1432–1437

[118] Guo Qi, Chen Tianshi, Zhou Zhihua, et al. Robust design space

modeling[J]. ACM Transactions on Design Automation of Electronic Systems, 2015, 20(2): 18: 1–18: 22

[119] Zhang Qianlong, Hou Rui, Yang Sibo, et al. The role of architecture simulators in the process of CPU design[J]. Journal of Computer Research and Development, 2019, 56(12): 2702−2719 (in Chinese)
(张乾龙, 侯锐, 杨思博, 等. 体系结构模拟器在处理器设计过程中的作用[J]. 计算机研究与发展, 2019, 56(12): 2702−2719)

[120] Hoste K, Eeckhout L. Microarchitecture-independent workload characterization[J]. IEEE Micro, 2007, 27(3): 63−72

[121] Jin Zhanpeng, Cheng A C. Evolutionary benchmark subsetting[J]. IEEE Micro, 2008, 28(6): 20−36

[122] Jin Zhanpeng, Cheng A C. SubsetTrio: An evolutionary, geometric, and statistical benchmark subsetting framework[J]. ACM Transactions on Modeling and Computer Simulation, 2011, 21(3): 21: 1–21: 23

[123] Jin Zhanpeng, Cheng A C. Improve simulation efficiency using statistical benchmark subsetting: An implantbench case study[C]// Proc of the 45th Annual Design Automation Conf. New York: ACM, 2008: 970–973

[124] Lee C, Potkonjak M, Mangione-Smith W H. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems[C]//Proc of the 30th Annual Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 1997: 330–335

[125] Guthaus M R, Ringenberg J S, Ernst D, et al. MiBench: A free, commercially representative embedded benchmark suite[C]//Proc of the 4th Annual IEEE Int Workshop on Workload Characterization. Piscataway, NJ: IEEE, 2001: 3–14

[126] Standard Performance Evaluation Corporation. SPEC CPU2000[EB/OL]. (2007-06-07)[2023-12-01]. https://www.spec.org/cpu2000

[127] Standard Performance Evaluation Corporation. SPEC CPU2006[EB/OL]. (2023-01-06)[2023-12-01]. https://www.spec.org/cpu2006

[128] Bienia C, Kumar S, Singh J P, et al. The parsec benchmark suite: Characterization and architectural implications[C]//Proc of the 17th Int Conf on Parallel Architectures and Compilation Techniques. New York: ACM, 2008: 72–81

[129] Woo S C, Ohara M, Torrie E, et al. The splash−2 programs: Characterization and methodological considerations[C]//Proc of the 22nd Annual Int Symp on Computer architecture. New York: ACM, 1995: 24–36

[130] Chandra D, Guo Fei, Kim S, et al. Predicting inter-thread cache contention on a chip multi-processor architecture[C]//Proc of the 11th Int Symp on High-Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2005: 340–351

[131] Hsu W C, Chen H, Yew P C, et al. On the predictability of program behavior using different input data sets[C]//Proc of the 6th Annual Workshop on Interaction between Compilers and Computer Architectures. Los Alamitos, CA: IEEE Computer Society, 2002: 45–53

[132] Hoste K, Eeckhout L. Comparing benchmarks using key microarchitecture-independent characteristics[C]//Proc of the 2nd IEEE Int Symp on Workload Characterization. Los Alamitos, CA: IEEE Computer Society, 2006: 83–92

[133] Yi J J, Sendag R, Eeckhout L, et al. Evaluating benchmark subsetting approaches[C]//Proc of the 2nd IEEE Int Symp on Workload Characterization. Los Alamitos, CA: IEEE Computer Society, 2006: 93–104

[134] Conte T M, Hirsch M A, Menezes K N. Reducing state loss for effective trace sampling of superscalar processors[C]//Proc of the 14th Int Conf on Computer Design. Los Alamitos, CA: IEEE Computer Society, 1996: 468–477

[135] Patil H, Cohn R, Charney M, et al. Pinpointing representative portions of large Intel® Itanium® programs with dynamic instrumentation[C]//Proc of the 37th Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 2004: 81–92

[136] Nair A A, John L K. Simulation points for SPEC CPU 2006[C]//Proc of the 26th Int Conf on Computer Design. Los Alamitos, CA: IEEE Computer Society, 2008: 397–403

[137] Lau J, Perelman E, Calder B. Selecting software phase markers with code structure analysis[C]//Proc of the 4th Int Symp on Code Generation and Optimization. Los Alamitos, CA: IEEE Computer Society, 2006: 135–146

[138] Lahiri K, Kunnoth S. Fast IPC estimation for performance projections using proxy suites and decision trees[C]//Proc of the 2017 IEEE Int Symp on Performance Analysis of Systems and Software. Los Alamitos, CA: IEEE Computer Society, 2017: 77–86

[139] Carlson T E, Heirman W, Eeckhout L. Sampled simulation of multi-threaded applications[C]//Proc of the 2013 IEEE Int Symp on Performance Analysis of Systems and Software. Los Alamitos, CA: IEEE Computer Society, 2013: 2–12

[140] Patil H, Pereira C, Stallcup M, et al. PinPlay: A framework for deterministic replay and reproducible analysis of parallel programs[C]//Proc of the 8th Annual IEEE/ACM Int Symp on Code Generation and Optimization. New York: ACM, 2010: 2–11

[141] Patil H, Isaev A, Heirman W, et al. ELFies: Executable region checkpoints for performance analysis and simulation[C]//Proc of the 19th IEEE/ACM Int Symp on Code Generation and Optimization. Piscataway, NJ: IEEE, 2021: 126–136

[142] Wenisch T F, Wunderlich R E, Falsafi B, et al. TurboSMARTS: Accurate microarchitecture simulation sampling in minutes[J]. ACM SIGMETRICS Performance Evaluation Review, 2005, 33(1): 408−409

[143] Khan T M, Pérez D G, Temam O. Transparent sampling[C]//Proc of the 10th Int Conf on Embedded Computer Systems: Architectures, Modeling and Simulation. Piscataway, NJ: IEEE, 2010: 28–36

[144] Eeckhout L, Luo Yue, De Bosschere K, et al. BLRL: Accurate and efficient warmup for sampled processor simulation[J]. The Computer Journal, 2005, 48(4): 451−459

[145] Haskins J W, Skadron K. Accelerated warmup for sampled microarchitecture simulation[J]. ACM Transactions on Architecture and Code Optimization, 2005, 2(1): 78−108

[146] Van Ertvelde L, Hellebaut F, Eeckhout L. Accurate and efficient cache warmup for sampled processor simulation through NSL–BLRL[J]. The Computer Journal, 2008, 51(2): 192−206

[147] Jiang Chuntao, Yu Zhibin, Jin Hai, et al. Shorter on-line warmup for sampled simulation of multi-threaded applications[C]//Proc of the 44th Int Conf on Parallel Processing. Los Alamitos, CA: IEEE Computer Society, 2015: 350–359

[148] Bell R, Eeckhout L, John L, et al. Deconstructing and improving statistical simulation in HLS[C]//Proc of the 2004 Workshop on Duplicating, Deconstructing and Debunking held in Conjunction with

the 31st Annual Int Symp on Computer Architecture. New York: ACM, 2004: 2−12

[149] Joshi A, Yi J J, Bell R H, et al. Evaluating the efficacy of statistical simulation for design space exploration[C]//Proc of the 2006 IEEE Int Symp on Performance Analysis of Systems and Software. Los Alamitos, CA: IEEE Computer Society, 2006: 70–79

[150] Eeckhout L, Bell R H, Stougie B, et al. Control flow modeling in statistical simulation for accurate and efficient processor design studies[C]//Proc of the 31st Annual Int Symp on Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2004: 350–361

[151] Bell R H, Bhatia R R, John L K, et al. Automatic testcase synthesis and performance model validation for high performance PowerPC processors[C]//Proc of the 2006 IEEE Int Symp on Performance Analysis of Systems and Software. Los Alamitos, CA: IEEE Computer Society, 2006: 154–165

[152] Lee H R, Sánchez D. Datamime: Generating representative benchmarks by automatically synthesizing datasets[C]//Proc of the 55th IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2022: 1144–1159

[153] Joshi A, Eeckhout L, Bell R H, et al. Performance cloning: A technique for disseminating proprietary applications as benchmarks[C]//Proc of the 2nd IEEE Int Symp on Workload Characterization. Los Alamitos, CA: IEEE Computer Society, 2006: 105–115

[154] Joshi A M, Eeckhout L, John L K, et al. Automated microprocessor stressmark generation[C]//Proc of the 14th Int Symp on High Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2008: 229–239

[155] Joshi A, Eeckhout L, Bell R H, et al. Distilling the essence of proprietary workloads into miniature benchmarks[J]. ACM Transactions on Architecture and Code Optimization, 2008, 5(2): 10: 1–10: 33

[156] Ganesan K, John L K. Automatic generation of miniaturized synthetic proxies for target applications to efficiently design multicore processors[J]. IEEE Transactions on Computers, 2014, 63(4): 833−846

[157] Deniz E, Sen A, Kahne B, et al. MINIME: Pattern-aware multicore benchmark synthesizer[J]. IEEE Transactions on Computers, 2015, 64(8): 2239−2252

[158] Lee K, Evans S, Cho S. Accurately approximating superscalar processor performance from traces[C]//Proc of the 2009 IEEE Int Symp on Performance Analysis of Systems and Software. Los Alamitos, CA: IEEE Computer Society, 2009: 238–248

[159] Lee K, Cho S. In-N-Out: Reproducing out-of-order superscalar processor behavior from reduced in-order traces[C]//Proc of the 19th Annual Int Symp on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems. Los Alamitos, CA: IEEE Computer Society, 2011: 126–135

[160] Lee K, Cho S. Accurately modeling superscalar processor performance with reduced trace[J]. Journal of Parallel and Distributed Computing, 2013, 73(4): 509−521

[161] Ganesan K, Jo J, John L K. Synthesizing memory-level parallelism aware miniature clones for SPEC CPU2006 and ImplantBench workloads[C]//Proc of the 2010 IEEE Int Symp on Performance Analysis of Systems & Software. Los Alamitos, CA: IEEE Computer Society, 2010: 33–44

[162] Panda R, Zheng Xinnian, John L K. Accurate address streams for LLC and beyond (SLAB): A methodology to enable system exploration[C]//Proc of the 2017 IEEE Int Symp on Performance Analysis of Systems and Software. Los Alamitos, CA: IEEE Computer Society, 2017: 87–96

[163] Van Biesbrouck M, Sherwood T, Calder B. A co-phase matrix to guide simultaneous multithreading simulation[C]//Proc of the 2004 IEEE Int Symp on Performance Analysis of Systems and Software. Los Alamitos, CA: IEEE Computer Society, 2004: 45–56

[164] Yi J J, Kodakara S V, Sendag R, et al. Characterizing and comparing prevailing simulation techniques[C]//Proc of the 11th Int Symp on High-Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2005: 266–277

[165] Tairum Cruz M, Bischoff S, Rusitoru R. Shifting the barrier: Extending the boundaries of the barrierpoint methodology[C]//Proc of the 2018 IEEE Int Symp on Performance Analysis of Systems and Software. Los Alamitos, CA: IEEE Computer Society, 2018: 120–122

[166] Bell R H, John L K. Efficient power analysis using synthetic testcases[C]//Proc of the 1st IEEE Int Symp Workload Characterization. Piscataway, NJ: IEEE, 2005: 110–118

[167] Penry D A, Fay D, Hodgdon D, et al. Exploiting parallelism and structure to accelerate the simulation of chip multi-processors[C]//Proc of the 12th Int Symp on High-Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2006: 29–40

[168] Mariani G, Palermo G, Zaccaria V, et al. DeSpErate: Speeding-up design space exploration by using predictive simulation scheduling[C/OL]//Proc of the 17th Design, Automation & Test in Europe Conf & Exhibition. Piscataway, NJ: IEEE, 2014[2023-12-18]. https://ieeexplore.ieee.org/document/6800432?arnumber=6800432

[169] Li Bin, Peng Lu, Ramadass B. Accurate and efficient processor performance prediction via regression tree based modeling[J]. Journal of Systems Architecture, 2009, 55(10): 457−467

[170] Pang Jiufeng, Li Xiafeng, Xie Jinsong, et al. Microarchitectural design space exploration via support vector machine[J]. Acta Scientiarum Naturalium Universitatis Pekinensis, 2010, 46(1): 55−63

[171] Cook H, Skadron K. Predictive design space exploration using genetically programmed response surfaces[C]//Proc of the 45th Annual Design Automation Conf. New York: ACM, 2008: 960–965

[172] Zhai Jianwang, Bai Chen, Zhu Binwu, et al. McPAT-Calib: A microarchitecture power modeling framework for modern CPUs[C/OL]//Proc of the 40th IEEE/ACM Int Conf on Computer Aided Design. Piscataway, NJ: IEEE, 2021[2023-12-18]. https://ieeexplore.ieee.org/document/9643508

[173] Zhai Jianwang, Bai Chen, Zhu Binwu, et al. McPAT-calib: A RISC-V boom microarchitecture power modeling framework[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023, 42(1): 243−256

[174] Givargis T, Vahid F, Henkel J. System-level exploration for Pareto-optimal configurations in parameterized systems-on-a-chip[C]//Proc

of the 20th IEEE/ACM Int Conf on Computer Aided Design. Los Alamitos, CA: IEEE Computer Society, 2001: 25–30

[175] Yazdani R, Sheidaeian H, Salehi M E. A fast design space exploration for VLIW architectures[C]//Proc of the 22nd Iranian Conf on Electrical Engineering. Piscataway, NJ: IEEE, 2014: 856–861

[176] Kansakar P, Munir A. A design space exploration methodology for parameter optimization in multicore processors[J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(1): 2−15

[177] Ascia G, Catania V, Di Nuovo A G, et al. Performance evaluation of efficient multi-objective evolutionary algorithms for design space exploration of embedded computer systems[J]. Applied Soft Computing, 2011, 11(1): 382−398

[178] Mariani G, Palermo G, Silvano C, et al. An efficient design space exploration methodology for multi-cluster VLIW architectures based on artificial neural networks[C]//Proc of the 16th IFIP/IEEE Int Conf on Very Large Scale Integration. Piscataway, NJ: IEEE, 2008: 13−15

[179] Zaccaria V, Palermo G, Castro F, et al. MULTICUBE Explorer: An open source framework for design space exploration of chip multi-processors[C]//Proc of the 23rd Int Conf on Architecture of Computing Systems. Hannover, Germany: VDE Verlag, 2010: 325–331

[180] Mariani G, Brankovic A, Palermo G, et al. A correlation-based design space exploration methodology for multi-processor systems-on-chip[C]//Proc of the 47th Design Automation Conf. New York: ACM, 2010: 120–125

[181] Wang Duo, Yan Mingyu, Liu Xin, et al. A high-accurate multi-objective exploration framework for design space of CPU[C/OL]// Proc of the 60th ACM/IEEE Design Automation Conf. Piscataway, NJ: IEEE, 2023[2023-12-18]. https://ieeexplore.ieee.org/document/10247790

[182] Wang Duo, Yan Mingyu, Teng Yihan, et al. A high-accurate multi-objective ensemble exploration framework for design space of CPU microarchitecture[C]//Proc of the 33rd Great Lakes Symp on VLSI 2023. New York: ACM, 2023: 379–383

[183] Beltrame G, Fossati L, Sciuto D. Decision-theoretic design space exploration of multiprocessor platforms[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2010, 29(7): 1083−1095

[184] Beltrame G, Nicolescu G. A multi-objective decision-theoretic exploration algorithm for platform-based design[C]//Proc of the 14th Design, Automation & Test in Europe Conf & Exhibition. Piscataway, NJ: IEEE, 2011: 1192−1195

[185] Sheldon D, Vahid F, Lonardi S. Soft-core processor customization using the design of experiments paradigm[C]//Proc of the 10th Design, Automation & Test in Europe Conf & Exhibition. Piscataway, NJ: IEEE, 2007: 821−826

[186] Mariani G, Palermo G, Silvano C, et al. Meta-model assisted optimization for design space exploration of multi-processor systems-on-chip[C]//Proc of the 12th Euromicro Conf on Digital System Design, Architectures, Methods and Tools. Los Alamitos, CA: IEEE Computer Society, 2009: 383–389

[187] Palermo G, Silvano C, Zaccaria V. Multi-objective design space exploration of embedded systems[J]. Journal of Embedded Computing, 2005, 1(3): 305−316

[188] Wu Nan, Xie Yuan, Hao Cong. IronMan: GNN-assisted design space exploration in high-level synthesis via reinforcement learning[C]// Proc of the 31st Great Lakes Symp on VLSI. New York: ACM, 2021: 39–44

[189] Wu Nan, Xie Yuan, Hao Cong. IronMan-Pro: Multiobjective design space exploration in HLS via reinforcement learning and graph neural network-based modeling[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023, 42(3): 900−913

[190] Kao S C, Jeong G, Krishna T. ConfuciuX: Autonomous hardware resource assignment for DNN accelerators using reinforcement learning[C]//Proc of the 53rd Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2020: 622–636

[191] Feng Lang, Liu Wenjian, Guo Chuliang, et al. GANDSE: Generative adversarial network based design space exploration for neural network accelerator design[J]. ACM Transactions on Design Automation of Electronic Systems, 2023, 28(3): 35: 1−35: 20

[192] Akram A, Sawalha L. A survey of computer architecture simulation techniques and tools[J]. IEEE Access, 2019, 7: 78120−78145

[193] Manjikian N. Multiprocessor enhancements of the simplescalar tool set[J]. SIGARCH Computer Architecture News, 2001, 29(1): 8−15

[194] Qureshi Y M, Simon W A, Zapater M, et al. Gem5-X: A many-core heterogeneous simulation platform for architectural exploration and optimization[J]. ACM Transactions on Architecture and Code Optimization, 2021, 18(4): 44: 1–44: 27

[195] Carlson T E, Heirman W, Eyerman S, et al. An evaluation of high-level mechanistic core models[J]. ACM Transactions on Architecture and Code Optimization, 2014, 11(3): 28: 1–28: 25

[196] Tan Zhangxi, Waterman A, Cook H, et al. A case for fame: FPGA architecture model execution[C]//Proc of the 37th Annual Int Symp on Computer Architecture. New York: ACM, 2010: 290–301

[197] Lee Y, Waterman A, Cook H, et al. An agile approach to building RISC-V microprocessors[J]. IEEE Micro, 2016, 36(2): 8−20

[198] Di Biagio A, Davis M. llvm-mca: A static performance analysis tool[EB/OL]. (2018−03−01)[2023-12-01]. https://lists.llvm.org/pipermail/llvm-dev/2018-March/121490.html

[199] Mendis C, Renda A, Amarasinghe D S, et al. Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks[C]//Proc of the 36th Int Conf on Machine Learning. New York: PMLR, 2019: 4505–4515

[200] Blocklove J, Garg S, Karri R, et al. Chip-Chat: Challenges and opportunities in conversational hardware design[C/OL]//Proc of the 5th ACM/IEEE Workshop on Machine Learning for CAD. Piscataway, NJ: IEEE, 2023[2023-12-18]. https://ieeexplore.ieee.org/document/10299874

[201] Chang Kaiyan, Wang Ying, Ren Haimeng, et al. ChipGPT: How far are we from natural language hardware design[J]. arXiv preprint, arXiv: 2305.14019, 2023

[202] Lu Yao, Liu Shang, Zhang Qijun, et al. RTLLM: An open-source benchmark for design RTL generation with large language model[J]. arXiv preprint, arXiv: 2308.05345, 2023

[203] Balkind J, Chang Tingjung, Jackson P J, et al. OpenPiton at 5: A

nexus for open and agile hardware design[J]. IEEE Micro, 2020, 40(4): 22−31

[204] Bachrach J, Vo H, Richards B, et al. Chisel: Constructing hardware in a scala embedded language[C]//Proc of the 49th Annual Design Automation Conf. New York: ACM, 2012: 1216–1225

[205] Patel H D, Shukla S K. Tackling an abstraction gap: Co-simulating SystemC DE with Bluespec ESL[C]//Proc the of 10th Design, Automation & Test in Europe Conf & Exhibition. Piscataway, NJ: IEEE, 2007: 279−284

[206] Bourgeat T, Pit-Claudel C, Chlipala A, et al. The essence of Bluespec: A core language for rule-based hardware design[C]//Proc of the 41st ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2020: 243–257

[207] Käyrä M, Hämäläinen T D. A survey on system-on-a-chip design using Chisel HW construction language[C/OL]//Proc of the 47th Annual Conf of the IEEE Industrial Electronics Society. Piscataway, NJ: IEEE, 2021[2023-12-18]. https://ieeexplore.ieee.org/document/9589614

[208] Wang Kaifan, Xu Yinan, Yu Zihao, et al. XiangShan open-source high performance RISC-V processor design and implementation[J]. Journal of Computer Research and Development, 2023, 60(3): 476−493(in Chinese)
(王凯帆, 徐易难, 余子濠, 等. 香山开源高性能 RISC-v 处理器设计与实现[J]. 计算机研究与发展, 2023, 60(3): 476−493)

[209] Lee B C, Brooks D M, Supinski B R de, et al. Methods of inference and learning for performance modeling of parallel applications[C]//Proc of the 12th ACM SIGPLAN Sympon Principles and Practice of Parallel Programming. New York: ACM, 2007: 249–258

[210] Hallschmid P, Saleh R. Fast design space exploration using local regression modeling with application to ASIPs[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2008, 27(3): 508−515

[211] Zhang Changshu, Ravindran A, Datta K, et al. A machine learning approach to modeling power and performance of chip multiprocessors[C]//Proc of the 29th Int Conf on Computer Design. Los Alamitos, CA: IEEE Computer Society, 2011: 45–50

[212] Beg A, Prasad P W C, Singh A K, et al. A neural model for processor-throughput using hardware parameters and software's dynamic behavior[C]//Proc of the 12th Int Conf on Intelligent Systems Design and Applications. Piscataway, NJ: IEEE, 2012: 821–825

[213] Paone E, Vahabi N, Zaccaria V, et al. Improving simulation speed and accuracy for many-core embedded platforms with ensemble models[C]//Proc of the 16th Design, Automation & Test in Europe Conf & Exhibition. Piscataway, NJ: IEEE, 2013: 671–676

[214] Castillo P A, Mora A M, Guervós J J M, et al. Architecture performance prediction using evolutionary artificial neural networks[C]//Proc of the Applications of Evolutionary Computing. Berlin: Springer, 2008: 175–183

[215] Khan S, Xekalakis P, Cavazos J, et al. Using predictive modeling for cross-program design space exploration in multicore systems[C]//Proc of the 16th Int Conf on Parallel Architecture and Compilation Techniques. Los Alamitos, CA: IEEE Computer Society, 2007: 327–338

[216] Dubach C, Jones T M, O'Boyle M F P. Rapid early-stage microarchitecture design using predictive models[C]//Proc of the 27th Int Conf on Computer Design. Los Alamitos, CA: IEEE Computer Society, 2009: 297–304

[217] Özisikyilmaz B, Memik G, Choudhary A N. Machine learning models to predict performance of computer system design alternatives[C]//Proc of the 37th Int Conf on Parallel Processing. Los Alamitos, CA: IEEE Computer Society, 2008: 495–502

[218] Özisikyilmaz B, Memik G, Choudhary A N. Efficient system design space exploration using machine learning techniques[C]//Proc of the 45th Design Automation Conf. New York: ACM, 2008: 966–969

[219] Ghosh A, Givargis T. Cache optimization for embedded processor cores: An analytical approach[J]. ACM Transactions on Design Automation of Electronic Systems, 2004, 9(4): 419−440

[220] Li Sheng, Chen Ke, Ahn J H, et al. CACTI-p: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques[C]//Proc of the 30th Int Conf on Computer-Aided Design. Los Alamitos, CA: IEEE Computer Society, 2011: 694–701

[221] Li Sheng, Ahn J H, Strong R D, et al. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures[C]//Proc of the 42nd Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2009: 469–480

[222] Karkhanis T S, Smith J E. A first-order superscalar processor model[C]//Proc of the 31st Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2004: 338–349

[223] Genbrugge D, Eyerman S, Eeckhout L. Interval simulation: Raising the level of abstraction in architectural simulation[C/OL]//Proc of the 16th Int Symp on High-Performance Computer Architecture. Piscataway, NJ: IEEE, 2010[2023-12-18]. https://ieeexplore.ieee.org/document/5416636

[224] Breughe M, Eyerman S, Eeckhout L. A mechanistic performance model for superscalar in-order processors[C]//Proc of the 2012 IEEE Int Symp on Performance Analysis of Systems & Software. Los Alamitos, CA: IEEE Computer Society, 2012: 14–24

[225] Van den Steen S, Eyerman S, De Pestel S, et al. Analytical processor performance and power modeling using micro-architecture independent characteristics[J]. IEEE Transactions on Computers, 2016, 65(12): 3537−3551

[226] De Pestel S, Van den Steen S, Akram S, et al. RPPM: Rapid performance prediction of multithreaded workloads on multicore processors[C]//Proc of the 2019 IEEE Int Symp on Performance Analysis of Systems and Software. Piscataway, NJ: IEEE, 2019: 257–267

[227] Jongerius R, Mariani G, Anghel A, et al. Analytic processor model for fast design-space exploration[C]//Proc of the 33rd IEEE Int Conf on Computer Design. Los Alamitos, CA: IEEE Computer Society, 2015: 411–414

[228] Jongerius R, Anghel A, Dittmann G, et al. Analytic multi-core processor model for fast design-space exploration[J]. IEEE Transactions on Computers, 2018, 67(6): 755−770

[229] Kwon J, Carloni L P. Transfer learning for design-space exploration with high-level synthesis[C]//Proc of the 2nd ACM/IEEE Workshop on Machine Learning for CAD. New York: ACM, 2020: 163–168

[230] Zhang Zheng, Chen Tinghuan, Huang Jiaxin, et al. A fast parameter tuning framework via transfer learning and multi-objective Bayesian optimization[C]//Proc of the 59th ACM/IEEE Design Automation Conf. New York: ACM, 2022: 133–138

[231] Zhang Keyi, Asgar Z, Horowitz M. Bringing source-level debugging frameworks to hardware generators[C]//Proc of the 59th ACM/IEEE Design Automation Conf. New York: ACM, 2022: 1171–1176

[232] Xiao Qingcheng, Zheng Size, Wu Bingzhe, et al. HASCO: Towards agile hardware and software co-design for tensor computation[C]//Proc of the 48th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2021: 1055–1068

[233] Esmaeilzadeh H, Ghodrati S, Kahng A B, et al. Physically accurate learning-based performance prediction of hardware-accelerated ML algorithms[C]//Proc of the 4th ACM/IEEE Workshop on Machine Learning for CAD. New York: ACM, 2022: 119–126

[234] Sun Qi, Chen Tinghuan, Liu Siting, et al. Correlated multi-objective multi-fidelity optimization for HLS directives design[C]//Proc of the 24th Design, Automation & Test in Europe Conf & Exhibition. Piscataway, NJ: IEEE, 2021: 46–51

[235] Wu Y N, Tsai P A, Parashar A, et al. Sparseloop: An analytical approach to sparse tensor accelerator modeling[C]//Proc of the 55th IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2022: 1377–1395

[236] Huang Qijing, Kang M, Dinh G, et al. CoSA: Scheduling by constrained optimization for spatial accelerators[C]//Proc of the 48th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2021: 554–566

[237] Mei Linyan, Houshmand P, Jain V, et al. ZigZag: Enlarging joint architecture-mapping design space exploration for DNN accelerators[J]. IEEE Transactions on Computers, 2021, 70(8): 1160−1174

Wang Duo, born in 1995. PhD. Student member of CCF. His main research interests include processor design space exploration and computer architecture.

王　铎, 1995 年生. 博士. CCF 学生会员. 主要研究方向为处理器设计空间探索、计算机体系结构.

Liu Jinglei, born in 1982. Master, senior engineer. His main research interests include computility network and computer architecture.

刘景磊, 1982 年生. 硕士, 高级工程师. 主要研究方向为算力网络、计算机体系结构.

Yan Mingyu, born in 1990. PhD, associate professor. Member of CCF. His main research interest includes graph based hardware accelerator and dataflow architecture.

严明玉, 1990 年生. 博士, 副研究员. CCF 会员. 主要研究方向为基于图的硬件加速器、数据流架构.

Teng Yihan, born in 2000. Master. His main research interests include graph-based hardware accelerator and high-throughput computer architecture.

滕亦涵, 2000 年生. 硕士. 主要研究方向为基于图的硬件加速器和高吞吐量计算体系结构.

Han Dengke, born in 1998. Master candidate. His main research interest includes graph-based hardware accelerator and high-throughput computer architecture.

韩登科, 1998 年生. 硕士研究生. 主要研究方向为基于图的硬件加速器和高吞吐量计算体系结构.

Ye Xiaochun, born in 1981. PhD, professor. Member of CCF. His main research interests include software simulation, algorithm paralleling and optimizing, and architecture for high performance computer.

叶笑春, 1981 年生. 博士, 研究员. CCF 会员. 主要研究方向为软件仿真、算法并行优化、高性能计算机架构.

Fan Dongrui, born in 1979. PhD, professor. Distinguished member of CCF. His main research interests include manycore processor design, high throughput processor design, and low power microarchitecture.

范东睿, 1979 年生. 博士, 研究员. CCF 杰出会员. 主要研究方向为众核处理器设计、高通量处理器设计、低功耗微架构.